KNOWLEDGE DISCOVERY: A NEURAL NETWORK APPROACH

By

HYEONCHEOL KIM

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1998

To my parents, Sang-Won Kim and Yong-Sook Shin

ACKNOWLEDGMENTS

I am very grateful to my advisor, Dr. LiMin Fu for his guidance and constant encouragement throughout my years at the University of Florida. I would also like to thank other members of my supervisory committee, Dr. Douglas D. Dankel, Dr. Gerhard X. Ritter, Dr. Baba C. Vemuri, and Dr. Jose C. Principe, for their kind understanding and willingness to serve on my supervisory committee.

My deepest respect and gratitude go to my parents for their wisdom and their enormous love and support. Everything that I am today is because of them. I thank my brother and relatives for their constant support and loving concern.

Finally, I thank all the friends I have made in Gainesville for their friendship and lovely memories that I will never forget.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

KNOWLEDGE DISCOVERY: A NEURAL NETWORK APPROACH

By

HYEONCHEOL KIM

August, 1998

Chairman: Dr. LiMin Fu
Major Department: Computer and Information Science and Engineering

Constructing a good model based on observed data and understanding the knowledge generated by the model are two key issues in knowledge discovery problems. The neural network is a good model that can estimate any smooth nonlinear function from the training data without any *a priori* assumptions. Recently, there has been much research on rule extraction from neural networks. One of the major problems investigated is reduction of rule search space. Several heuristics based on the KnowledgeTron algorithm have been introduced to reduce the search space. The algorithm can also be used to refine the neural network structure. In this thesis, a new search algorithm based on ordered attributes is introduced. It reduces search space dramatically and finds maximally general and confident rules for nodes in the network. In quantitative domains, a GA-based search algorithm is also proposed for rule extraction on the continuous-valued attributes. For a case study, industrial efforts in data mining and data warehousing are described, and customer churn analysis in the

telecommunications industry is used as a demonstration of the proposed algorithms for industrial application.

CHAPTER 1
INTRODUCTION

1.1 Knowledge Discovery and Neural Networks

Knowledge discovery is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [20]. A model is constructed based on the observed data, and the model helps us to *understand* what is happening and it may allow us to *predict* the outcome of similar phenomena.

Traditionally, statistical techniques have been used to construct models from observations. There are many techniques available, of which linear regression is probably the most used in practice. Linear regression and other statistical techniques work well for many situations. Generated models often have a good predictive accuracy. However, there are also some drawbacks in using statistical techniques on commercial databases:

- Complications occur when the relationship between two variables cannot be modeled by a linear function. There would be some other function, e.g. a polynomial function. But it is very difficult to know in advance which function fits your data base. One solution is to test many different functions, and see which fits the data best. Alternatively, one could resort to non-parametric regression, such as kernel smoothing, in which case the function is flexibly adapted to the data. This will, however, considerably decrease the insight that the model provides.

- Complex problems occur when multiple variables are involved. If the number of variables gets too large, however, linear regression does not work very well.

Consequently, you may need to reduce the number of variables in your model or use a dimension reduction technique such as Principal Component Analysis.

- Another problem concerns non-numerical, or categorical data. If you use these data in your numerical model, you must code them as numerical values. Coding introduces a bias into the model.

- Most statistical techniques make assumptions about the probability distribution of variables. In linear regression, for example, it is assumed that the variables are normally distributed. Since these exact conditions are seldom found in practice, models lose predictive accuracy.

Real-world databases typically contain many variables, some of which are categorical. The form of the relationships among these variables is often unknown. This makes the model more difficult to understand and its predictions less accurate.

Statistical techniques work best if you already have some idea about the model that you want to discover, i.e., if you know which of the variables are important and which functional form should be used to relate these variables to each other. For many real-life situations, however, the model is not known a priori.

A neural network is able to find any type of non-linear decision boundary. In terms of model evaluation, while networks of appropriate size can universally approximate any smooth function to any desired degree of accuracy, relatively little is known about the representation properties of fixed size networks estimated from finite data sets. The standard squared error and cross entropy loss functions used to train neural networks can be viewed as log-likelihood functions for regression and classification, respectively. Backpropagation is a parameter search method which performs gradient descent in parameter (i.e., connection weight) space to find a local maximum of the likelihood function starting from random initial conditions. Neural networks do not make any a priori assumptions about the functional relationships involved, but

flexibly adapt the function to the data used in training. This is a useful property if little is known about the phenomenon that one wants to model.

Nonlinear regression methods, though powerful in representational power, can be very difficult to interpret. In general, neural networks yield quite accurate models, but they provide little or no insight into the problem concerned. This model is hidden in a black box and can be used to *predict*, but not to *understand*, the environment. It is difficult to explain why a network makes particular predictions. For applications where one needs to understand the model, this may be the reason to use alternative techniques. However, models constructed for knowledge discovery and data mining should be expressed in a language that is understood easily by human users. This allows the user to inspect the knowledge generated by the model, and judge its usability for particular decisions.

Numerous studies on rule extraction from trained feed-forward neural networks have been conducted in recent years. Different approaches have been proposed with different justifications: some of the justifications of rule extraction include explanation capability, rule refinement [6, 32, 31, 33], knowledge acquisition for symbolic AI systems, generation of examples for symbolic learning algorithms [5], improvement in generalization of the network [9], knowledge discovery, and data exploration [25, 17].

Key issues of rule extraction methods involve:

Transparency. The Open-box approach extracts rules at the level of individual units within the networks and aggregates the rules from each unit to form the composite rule base for the neural network as a whole. The Black-box approach observes the input/output behavior of the network and extracts rules that map inputs directly into outputs.

Validity Test. Given a rule hypothesis, it is *valid* if it is true regardless of the values of attributes not referenced by the hypothesis. Validity testing technique is

important in terms of the quality of the rules and reduction of the complexity of hypothesis space to be searched.

Search Space. Given a hypothesis space defined, searching for maximally general and accurate rules is done in a *top-down* or *bottom-up* manner. The top-down approach begins with the most general hypothesis and searches down by adding attributes, while the bottom-up method begins with the most specific hypothesis (i.e., training instances) and searches up by dropping attributes one at a time.

Transparency. The open-box approach interprets units in a network as logical conceptual entities and connection weights between nodes as correlational information between the concepts. Thus, the approach has the following assumptions:

1. Units of the network are threshold units. This is done by applying hard-limiting-like activation functions (e.g., a sigmoid function with very steep slope) or by clustering activations of the units into a finite number of discrete values.

2. Training does not significantly alter the meaning of units. It is required because each unit in the network is considered as a logical concept.

The approach that requires the two assumptions above works successfully for rule refinement systems [6, 32, 31] where initial knowledge rules are incorporated into a neural network and refined rules are extracted from the network after training. However, it is not universally applicable to arbitrary backpropagation-style neural networks because it requires a particular structure, representation and special training procedure to work successfully. When applied to a general neural network, it may not provide an accurate representation of the network.

In general, the black-box approach does not assume that the networks given to it have any particular architecture, nor that they were trained in any special way.

Another concern is the computational complexity of the search. The open-box approach searches separate hypothesis spaces for each hidden and output unit, while the black-box method searches a space for each output class. Usually the size of each search space is exponential in the number of incoming attributes. Since many real-world problems involve a large number of features, this problem is significant. Some heuristics or search techniques are introduced to reduce the search space [8, 7, 9, 5].

Validity Test. Testing validity of a rule for a neural network is simple in the open-box approach if and only if the rules extracted from each individual unit are valid. Given the intermediate rules from individual conceptual units, logical aggregation does not interfere with the validity of the rules. Thus, the concern is the validity testing of the intermediate rules extracted from hidden and output units of the network. Given a combination of attributes at each unit's search space, the KT algorithm [7] uses heuristics that test the validity condition and reduce the search space. Consider a node, h_j, with n total incoming weights, $w_{ji} \in W$ (p positive weights w_{ji}^{pos} and q negative weights w_{ji}^{neg}), and a threshold θ_j. Thus, $t = p + q$. Given a combination (C) of positive attributes and negative attributes, if the following is true, then keep the combination as a rule:

$$(\sum_{i=1} w_{ji}^{pos} + \sum_{i=1} w_{ji}^{neg}) > \theta_j, \text{ where } (w_{ji}^{pos} \in C) \text{ and } (w_{ji}^{neg} \in W, \text{ but } \notin C). \quad (1.1)$$

Validity testing in a black-box approach differs in that it does not investigate connection weights, but it checks the sensitivity of output classes according to a rule to be tested. Thrun [29, 30] developed VIA (validity-interval analysis) for testing continuous-valued attributes.

Search Space. In general, hypothesis space is structured as a tree where each node represents a hypothesis (i.e., a rule candidate). The node at the top of the tree

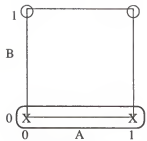
represents the most general rule (i.e., it covers all possible instances) and the nodes at the bottom level represent specific instances.

A key issue is how to find a set of maximally general and valid rules efficiently in the hypothesis space. The search is performed in *top-down* or *bottom-up* manner and stops when it finds a set of hypothesis nodes that satisfy the validity conditions. The top-down approach begins with the most general rule and searches down the space by adding attributes until the validity requirements are met [6, 8]. The bottom-up approach begins with specific instances of a training data set and searches up by dropping attributes until it violates the validity requirements [5].

1.2 Representation of Knowledge: IF-THEN rules

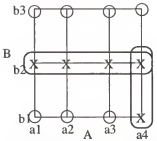
A rule has the form "*if the premise, then the conclusion.*" The premise is composed of a number of positive attributes (e.g., a_i) and negative attributes (e.g., *not* a_i), and so is the conclusion (e.g., c_i and *not* c_i). A rule is called a confirming rule if the conclusion is c, or a disconfirming rule if the conclusion is *not* c_i. In the basic form of a rule, the rule's premise is limited to a conjunction of attributes and the rule's conclusion is limited to a single attribute (i.e., a single class). Disjunction is not allowed in the premise and conjunction of multiple classes is not allowed in the conclusion. However, the presence of multiple rules with same conclusion represents disjunction. A rule with a conjunction of conclusions is represented by multiple rules with same premise but different conclusions.

The space that is covered by the rule is determined by the types of attributes in a problem domain. An attribute can be *binary*, *multi-valued* or *continuous-valued*. Figure 1.1 illustrates the rules with three different types of attributes in two-dimensional space. For the types of binary and multi-valued attributes, a rule covers points in the space while a rule with continuous-valued ones covers a squared space which does not



Binary attribute space

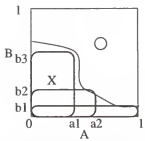
if not B, then X



Multi-valued attribute space

if (B=b2), then X

if (A=a4) and not (B=b3), then X



Continuous-valued attribute space

if (0<A<a1) and (0<B<b3), then X

if (0<A<a2) and (0<B<b2), then X

if (0<B<b1), then X

Figure 1.1. Three different types of attributes and their rules. Input space is divided by two concepts, O and X.

necessarily fit perfectly to the arbitrarily shaped space defined by a concept. Therefore, the *IF-THEN* rule does not guarantee to cover the problem domain completely.

In any case, a rule is represented in a string of attributes. For binary attribute rules, a rule r_i in a rule set R is a string of bits of 0, 1 or # where the # is a *don't-care* symbol. In a premise, positive attributes are 1's, negatives are 0's and the others are #'s. In a consequence, the positive attribute is 1 and the negative is 0. There is no # in a consequence. An instance in a data set can be represented in the same way. For example, we have 5 input attributes of a1, a2, a3, a4 and a5, and 3 output attributes of b1, b2 and b3, assuming that all the attributes are binary-valued, that is, *yes* or *no*. Consider a rule:

```
if (a1) and (a2) and (not a4) then (b2).
```

This rule is encoded in a string of schema form

```
11#0# 010.
```

Consider the case with attributes that are not binary. We transform the attributes into binary form because of its efficiency and convenience. For example, we have 2

attributes, F1 and F2, where F1 has 3 values (f11, f12, f13) and F2 has 4 values (f21, f22, f23, f24), and one output class C that has 2 values (c1, c2). This gives a new set of binary attributes f11, f12, f13, f21, f22, f23, f24, c1 and c2. Consider the following rule:

```
if (F1 is f12) and (F2 is f24), then (C is c1).
```

This rule can be rewritten according to the new binary attributes as follows:

```
if (f12) and (f24), then (c1).
```

In this case, we should keep in mind that there is a mutually exclusive relationship between the value elements of an attribute. Thus, if f11 is 1, f12 and f13 should be 0's, not #'s. This rule is encoded as follows:

```
010 0001 10.
```

Even for a continuous-valued attribute case, a rule can be represented in a string of attributes where the values are continuous. For example, we have 2 attributes, F1 and F2, where F1 and F2 are normalized to a range [0.0, 1.0]. Consider the rule:

```
if (0<F1<0.7), then (c1).
```

This rule is encoded into the following string of four digits, where each attribute needs two digits, one for a low bound and the other for an upper bound:

```
[0.0 0.7, # #,  0 1].
```

Quality of a rule is evaluated with a few criteria. First of all, a rule should be *valid*. A rule is said to be *valid* when it is true no matter what the # symbols in the rule are (i.e., 0 or 1). In terms of rule evaluation, the following two criteria should be considered: *accuracy* (specificity or goodness of fit) and *generality* (simplicity). Generality is about how often the left-hand side of a rule occurs. Accuracy is about

how often a rule is classified correctly, which is defined as the probability of its consequence being true given that its premise is matched among the test instances. When we evaluate a rule along with other rules in the system (context dependent evaluation), rules can be ranked according to how much they contribute to incorrect conclusions [35].

There have been some studies on information theory approaches to the measure of rule goodness. Smyth and Goodman [27, 28] proposed a quantity called the *J-measure*, which quantifies the information content of a rule or a hypothesis. They use a probabilistic rule defined as "*If (X=x) then (Y=y) with p.*" The probabilistic term $p(x)$ can be viewed as a criterion for generality in rules. For the measure of accuracy (goodness of fit), *cross entropy* is used. The *J-measure*, $J(Y; X = x)$, is the product of the two terms and maximizing it is equivalent to simultaneously maximizing the accuracy and generality. Piatetsky-Shapiro [20] proposed the use of $p(y)(p(x|y) - p(x))$ as a measure of rule goodness, which is based directly on probabilities. This measure is correlation-based while Smyth's *J-measure* is information-based useing a log-scale.

Kononenko and Bratko [14] pointed out the problem of *prior probability*. In a problem domain where the prior probability of one class is very high, even a naive classifier that groups all instances into the most likely class would achieve a high classification accuracy. Such high accuracy, achieved in a very simple-minded way in an easy domain, should not result in a more difficult domain. The evaluation criterion should, therefore, take into account the prior probabilities of classes.

When it comes to the evaluation of a set of rules, there might be inconsistency and incompleteness among the rules in the rule set. Several types include redundant rules, subsumed rules and conflicting rules. Conflicting rules include two situations:

an *overlapped example* refers to the one that is covered by more than one rule with different consequences, and an *unmatched example* is the one covered by no rules.

1.3 Overview

This research is intended to extract rules from trained neural networks efficiently. The focus is on the computational complexity of the rule search procedure, the quality of the extracted rules and a ruleset and application to well-known problems.

In Chapter 2, we view a neural network as a Bayesian classifier. Many issues concerning Bayesian probabilities are reviewed and their relationships with neural network training are discussed. Extracted rules are discussed in Bayesian terms, which provides better understanding of performance evaluation. We prove that a set of rules extracted from the Bayesian classifier always provides better performance than a default rule does.

In Chapter 3, decompositional approaches to rule extraction are reviewed. Fu's KT algorithm is used to show the basic concept of the approaches. Based on the KT algorithm, a few heuristic algorithms based on complexity controlling parameters are introduced, which reduce the extraction complexity without degrading the quality of rules. The KT algorithm with the proposed heuristics is applied to *network refinement*, a new approach to the neural network pruning problem.

In Chapter 4, the Ordered Attribute Search (OAS) algorithm is proposed. The algorithm is very efficient in terms of search complexity. Complexity comparison is performed on other algorithms and empirical experiments are performed on well-known data domains.

In Chapter 5, rule extraction on quantitative domains is discussed. A Genetic Algorithm approach to the problem is introduced.

In Chapter 6, as a case study, churn analysis in the telecommunications industry is done. This case study shows how the rule extraction procedure is applied to real-world data mining problems.

Chapter 7 presents a discussion of the preceding work along with conclusions.

CHAPTER 2
NEURAL NETWORKS AND RULES: A BAYESIAN VIEW

2.1 Introduction: Bayesian Classifier

In this chapter, a neural network and a ruleset are described as a Bayesian classifier. The Bayesian view provides the following probabilistic properties of the neural network and the extracted rules:

1. probabilistic rules,
2. generality of rules,
3. validity of rules and
4. minimum performance.

We also prove that a set of rules extracted from a Bayesian classifier gives a better performance than a Bayesian default decision rule.

In a Bayesian classifier, the distribution of inputs and target classes is assumed to be known exactly, and the prior probabilities of the classes are assumed to be known, so that the posterior probabilities can be computed by a simple application of Bayes theorem (or Bayes rule). The term *Bayes rule* is also used to mean any classification rule that gives results identical to those of a Bayes classifier. Bayes Rule (i.e., Bayesian Theorem) is defined as:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{\sum_{j=1}^{M} P(X|C_j)P(C_j)} \tag{2.1}$$

$$= \frac{P(X|C_i)P(C_i)}{P(X)}. \tag{2.2}$$

Bayes rule shows how observing the value of X changes the *a priori* probability $P(C_i)$ to the *a posteriori* probability $P(C_i|X)$. The $P(X)$ is common to all classes and is

considered a normalization factor that makes $\sum_{i=1}^{M} P(C_i|X) = 1$. $P(X|C_i)$ is the likelihood or conditional probability of producing the input X if the class is C_i. It is the degree to which training data reflect true likelihood distribution. Decision rule assigns X to the C_i with maximal a posteriori probability given X. That is, the decision class is determined by

$$P(C_d|X) = \max_{i=M}[P(C_i|X)] \tag{2.3}$$

where M is the number of classes. The decision rule (or classification rule) based on Bayes rule is *optimal*, giving the lowest expected error rate. If a priori probabilities $P(C_i)$ are equal, the decision rule says to choose the maximum-likelihood hypothesis C_{ML} that maximizes the likelihood $P(X|C_i)$.

Bayesian classifiers solve classification problems by calculating $P(C_i|X)$ for each class and assigning the input to the class with the highest Bayesian probability. However, it is not attainable because complete information is not known about the statistical distributions in each class, i.e., $P(X|C_i)$ and $P(C_i)$. There are two methods to supply the missing distribution information:

Parametric. This method makes assumptions about the nature of the distribution (for example, a Gaussian distribution) and then estimates the parameters of the distribution (for example, μ and σ for a Gaussian distribution).

Nonparametric. This method makes no assumptions about the specific distributions involved.

Conventional Bayesian classifiers use the *parametric* method to estimate the probabilities. One approach is to use the samples to estimate the unknown probabilities and probability densities and to use the resulting estimates as if they were the true values. The $P(C_i)$ is easily estimated from training samples. The $P(X|C_i)$ is hard to estimate if the number of training samples is small or the dimensionality of the input

vector X_i is large. Thus, general knowledge about the problem is used, assuming specific parametric distributions such as Gaussian or Gaussian mixture distribution. This simplifies the problem from one of estimating a function $P(X|C_i)$ to one of estimating the parameters μ and σ.

2.2 Neural Network as a Bayesian Classifier

A neural network is considered a *nonparametric* (distribution-free) technique that can be used without assuming that the forms of the underlying densities are known. It directly estimates *a posteriori* probabilities $P(C_i|X)$. Namely, the neural networks can be considered as a probabilistic distribution of $P(C_i|X)$, i.e.,

$$y_i(X) \approx P(C_i|X)$$

where X is an input vector, C_i is the i^{th} class and $y_i(X)$ is an output value of i^{th} output node given X.

Theoretical work on the relationship between neural-network classifiers and Bayes classification has been studied [21, 22, 34]. Wan [34] determined that minimizing an expected squared-error criteria is related to finding an optimal Bayesian classifier. Richard and Lippmann [21] proved that neural network outputs estimate Bayesian probabilities when desired outputs are *1 of M* and a squared-error or cross-entropy cost function is used. They also demonstrated that outputs of multilayer perceptron networks, trained with backpropagation, provide good estimates of Bayesian probabilities.

Let us define the following terms: Network desired output is $d_i = 1$ or 0; network actual output is $y_i(X) = \hat{P}(C_i|X)$, which is an estimate of true *a posteriori* probability $P(C_i|X)$; and normalized network output is $y_i^{norm}(X) = P(\hat{C}_i|X)$, which is defined as

$$P(\hat{C}_i|X) = \frac{\hat{P}(C_i|X)}{\sum_j \hat{P}(C_j|X)}. \tag{2.4}$$

A neural network trained with back propagation minimizes

$$\Delta = E\{\sum_{i=1}^{M}[y_i(X) - d_i]^2\}$$

so that the $y_i(X)$ estimates d_i. Richard and Lippmann [21] proved that minimizing Δ is reduced to minimizing $E\{\sum_{i=1}^{M}[y_i(X) - E(d_i|X)]^2\}$. As Wan [34] showed,

$$\begin{aligned} E(d_i|X) &= \textstyle\sum_{d_i \in \{0,1\}} d_i P(C_i|X) \\ &= P(d_i = 1|X) \\ &= P(C_i|X) \end{aligned} \tag{2.5}$$

so that $y_i(X)$ estimates $P(C_i|X)$. Thus, the decision rule requires the choice of C_i for which $y_i(X)$ is maximum.

When the estimation is accurate, network outputs can be treated as probabilities that sum to one, and the classification error rate is minimized. Accurate estimation is obtained only if

1. sufficient training data are available,

2. the network is complex enough and

3. classes are sampled with the true $P(C_i)$'s in the training data.

Accuracy of the estimation is calculated by comparing two probabilities: a posterior probability $P(C_i|X)$ and a normalized network output $y_i^{norm}(X) = P(\hat{C}_i|X)$. The Kullback-Leibler entropy $K(p, q)$, also called relative entropy or cross-entropy, is a measure of the distance between two distributions $p(x)$ and $q(x)$ and is defined as

$$K(p, q) = \sum_x p(x) \log(\frac{p(x)}{q(x)}). \tag{2.6}$$

The relative entropy is not a true distance because it is not symmetric, i.e.,

$$K(p, q) \neq K(q, p). \tag{2.7}$$

However, each expression in equation 2.7 can be interpreted as a quasi-distance that is always positive and is equal to zero if and only if $p(x) = q(x)$. Cross-entropy can be used to measure the distance between $P(C|X)$ and $P(\hat{C}|X)$:

$$K(P(C|X), P(\hat{C}|X)) = \sum_i P(C_i|X) \log(\frac{P(C_i|X)}{P(\hat{C}_i|X)}). \tag{2.8}$$

However, this is not a practical method because $P(C_i|X)$ should be known. Wan [34] proposed a more practical way of using the entropy of $P(\hat{C}_i|X)$:

$$H(P(\hat{C}|X)) = \sum_i P(\hat{C}_i|X) \log \frac{1}{P(\hat{C}_i|X)}. \tag{2.9}$$

This is a measure of the distribution's cross-entropy distance from the uniform distribution. H is maximum for the uniform distribution and the zero minimum if $P(\hat{C}_i|X) = 1$.

A conventional way to measure the generalization is to compare the frequency of classification errors in the training set to the frequency of errors of a test data set. This way provides information about the expected performance of the network relative to the performance on the training set, but not relative to the optimal classifier. The result should measure how closely the network has come to approximating the optimal Bayesian classifier. $E[K(P(C|X), P(\hat{C}|X))]$ accomplishes the measurement of how closely the neural network approximates the Bayesian classifier over all expected inputs. Because the $P(C|X)$ is not readily available, Wan [34] used Jensen's inequality to bound $E[K]$ as follows:

$$\begin{aligned} E[K(P(C|X), P(\hat{C}|X))] &\geq K(E[P(C|X)], E[P(\hat{C}|X)]) \\ &= K(P(C), P(\hat{C})). \end{aligned}$$

The $P(C)$ can be estimated from the training data by simply observing the frequency of occurrence of each class C_i in the training data

$$\frac{1}{T} \sum_{j=1}^{T} (d_i)_j \rightarrow E[d_i] = P(C_i). \tag{2.10}$$

Similarly, $P(\hat{C}|X)$ can be estimated by averaging the normalized outputs of each output neuron over all training samples

$$\frac{1}{T}\sum_{j=1}^{T} P(\hat{C}_i|X_j = X) \rightarrow E[P(\hat{C}_i|X)] = P(\hat{C}_i). \tag{2.11}$$

Thus, $K(P(C), P(\hat{C}))$ can be computed from the training data and is a measure of the distance between the true class random variable C and the network's estimate $\hat{C}$. $K(P(C), P(\hat{C}))$ is zero when $P(C)$ equals $P(\hat{C})$.

2.3 A Priori Information

The importance of a priori information is often overlooked. The Bayesian view of training data sets informs how improper a priori probabilities can mislead the estimation of classifiers and performance evaluation.

The decision rule selects a class C_1 in two class classification problems when

$$\begin{aligned} P(C_1|X) > P(C_2|X) &\equiv P(X|C_1)P(C_1) > P(X|C_2)P(C_2) \\ &\equiv P(X|C_1) > P(X|C_2)\frac{P(C_2)}{P(C_1)}. \end{aligned}$$

If $P(C_2) \ll P(C_1)$, then almost always the C_1 will be selected for a decision, thereby excluding the class C_2. In a speech application involving approximately 20,000 training samples, no samples were classified into classes with low a priori probability after training by a backpropagation network [3]. Barnard [2] showed experimental results about how a priori probabilities affect the performance of neural networks and Bayesian classifiers, given Gaussian a prior information. Generally a posteriori probability $P(C_i|X)$ is in proportion to a priori probability $P(C_i)$. However, if $P(C_i)$ is very low and overlapped partially with other classes, the class C_i is less likely to be selected, i.e., $P(C_i|X) < P(C_i)$. Classification performance of C_i reaches minimum near $P(C_i) = 0.5$.

Customary evaluation criteria, such as the relative frequency of correct classifications, do not account the difference between a priori probabilities of classes. Consider

a class C_1 whose a priori probability is 0.8. Classification accuracy of 80% easily is achieved because selecting the class C_1 for any inputs will give 80% of classification accuracy. This also is seen by expectation $E(P(C_1|X)) = P(C_1)$ given enough number of training samples. A classifier trained on the training samples should should produce classification accuracy of minimum 80%. Consider another data set whose highest class a priori probability is 0.2. 20% classification accuracy will occur even by rough guess. Suppose the first classifier gives a 75% classification accuracy and the latter is 30% accurate. If we use classification accuracy as a performance measure, the first classifier is better than the latter, but that is not necessarily true. Thus, a priori probabilities must be accounted for in performance measurement. Kononenko and Bratko [14] proposed an evaluation criterion for classifier systems other than classification accuracy that would evaluate information score of a classifier's answer to exclude the influence of a priori probabilities.

Consider a classification problem that classifies a person based on height into two classes, M and F for male and female. A binary attribute, $height > 170cm$, is used. True a priori probabilities $P(M)$ and $P(F)$ are 0.5 each because half of the population is female. Suppose $P(height > 170cm|M) = 0.7$ and $P(height > 170cm|F) = 0.1$ each. The training data set should reflect the true probabilities. Thus, $P(M|height > 170cm) = 0.875$ and $P(F|height > 170cm) = 0.175$ according to the Bayes theorem defined in equation 2.2. Namely, if a person's height is taller than $170cm$, then the person is a male with a 87.5% of probability. If the a priori probabilities of the training data set do not represent true probabilities, then they will cause a wrong estimation of a posteriori probability. For example, suppose that a classifier were trained on a training data set that includes 10 males, 70% of whom are taller than $170cm$, and 90 females, 10% of the person is taller than $170cm$. Then,

if a person's height is taller than $170cm$, it will be, female with a probability of 0.5625 under Bayes rule (2.2), an obviously wrong result.

Under Bayesian theory, minimum classification performance of a system should be

$$P(C_d) = \max_{i=1...M}[P(C_i)]$$

when true a priori class probabilities are given. For example, if the prior probability of the majority class C_d is 80%, the classification performance should be minimum of 80% by the default rule: *Any instance is classified as* C_d. Many methods do not provide a minimum classification performance for the resulting rule-based system. Hence, there are some cases that classification performance is worse than the default performance [14]. This is because prior class probabilities are not accounted for in the classifier system.

2.4 Rule Set as a Bayesian Classifier

With Bayesian probabilities available, there are the following three ways to obtain rules:

Default rule: When we have a priori class information only, it is always possible to use the *default rule*: *any instance is classified as class* C_d, *where* $P(C_d) = \max_{i=1...M}[P(C_i)]$, *irrespective of the attribute of the instance.* This default or no-data rule may even be adopted in practice if the cost of gathering data is too high.

Rule by likelihood ratio: Suppose we are able to observe data X on an individual, and that we know the probability distribution of X within each class C_i to be $P(X|C_i)$. Then for any two classes C_1 and C_2, the *likelihood ratio* $P(X|C_1)/P(X|C_2)$ provides the theoretical optimal form for discriminating the classes on the basis of data X.

Rule by posterior probability: When $P(C_d|X)$ is maximum among $P(C_i|X)$s $i = 1 \ldots M$, it is interpreted as a *probabilistic rule* such as

if X occurs, then C_d with probability of $P(C_d|X)$.

The posterior probabilities are obtained from a priori information or a large number of training data samples.

The input variable X of length l belongs to one of the following three types:

1. Instance X: An instance is one observation.

2. Hypothesis H: A hypothesis accounts for a set of instances. A string of l bits of 0's, 1's or $*$'s (*don't care's*). This hypothesis covers v^{l-d} instances when the size of H is d.

3. Domain D: A domain covers all possible instances in a problem domain. A string of l bits of $*$'s (*don't care's*), which covers total v^l instances.

Rule induction finds the hypotheses H such that $P(C_d|X_j)$ is the maximum among $P(C_i|X_j), i = 1 \ldots M$ for all X_j's such as $P(H|X_j) = 1$.

Given an instance input X, the Bayesian decision rule assigns X to the class C_d, which has the maximum class posterior probability among other classes, i.e.,

$$P(C_d|X) = \max_{i=1 \ldots M} P(C_i|X)$$

where M is the number of classes.

We define a *hypothesis* H as a vector whose elements are either 0, 1 or $*$ (i.e., *don't-care* symbol). Suppose that the size of an input vector is n and the one of H is l (i.e., the H has l non-"$*$" symbols). Then, H covers 2^{n-l} different instances X_j's for $j = 1 \ldots 2^{n-l}$.

The generality of H can be measured as

$$P(H) = \sum_{i=1}^{2^{n-l}} P(X_i). \tag{2.12}$$

The $P(X_i)$ and the generality $P(H)$ can be estimated from domain data as

$$P(X_i) \approx \frac{N_{X_i}}{N_D} \text{ and} \tag{2.13}$$

$$P(H) \approx \frac{N_H}{N_D} = \frac{\sum_i N_{X_i}}{N_D} \tag{2.14}$$

where N_{X_i} is the number of occurrences of X_i in the domain, N_H is the number of instances covered by H in the domain, and N_D is the number of total instances in the domain. Similarly, class prior probability $P(C_i)$ is estimated as

$$\begin{aligned} P(C_i) &\approx \frac{N_{C_i}}{N_D} \\ &= \frac{N_{C_i}}{\sum_{j=1}^{M} N_{C_j}}. \end{aligned}$$

Bayesian techniques provide good information in evaluating hypothetical rules and in manipulating the uncertainties of hypothetical rules. The hypothetical rule pairs a hypothesis and its decision class. Given a hypothesis, conventional rule extraction methods assign it to every class and evaluate the hypothetical rule (i.e., a hypothesis and its class) to decide the right class for the hypothesis. The Bayesian decision rule directly assigns the optimal decision class to the hypothesis instead of evaluating it for every class. Under the Bayesian decision rule, H is assigned to the class C_d such as

$$P(C_d|H) = \max_{i=1...M} P(C_i|H) = \max_{i=1...M} \frac{P(C_i \cap H)}{P(H)} \tag{2.15}$$

which also measures the certainty of the hypothetical rule. Thus, given a hypothetical rule composed of H and its optimal decision class C_d, the posterior probability can be estimated by the numbers of positive and negative examples of H defined as follows:

$$N_H^{pos} = \max_{j=1...M} N_{H \cap C_j} = N_{H \cap C_d} \tag{2.16}$$

$$N_H^{neg} = N_H - N_H^{pos} \tag{2.17}$$

$$P(C_d|H) \approx \frac{N_{H \cap C_d}}{N_H} = \frac{N_H^{pos}}{N_H}. \tag{2.18}$$

The definition 2.18 is considered as an accuracy or performance of H.

A hypothetical rule is called valid if it does not involve any negative examples, i.e., if it satisfies the following condition:

$$P(C_d|X_j) = \max_{i=1...M} P(C_i|X_j) \text{ for all } X_j's \text{ covered by } H. \tag{2.19}$$

The certainty of a hypothetical rule $P(C_d|H)$ is maximized if the rule is valid. Thus, a set of hypotheses is found , which are valid and maximally general over a problem domain, by satisfying the following two criteria:

1. $P(H)$ is maximized and

2. Equation 2.19 is satisfied.

The fact that decision classes of hypotheses are determined by the Bayesian decision rule gives many useful and robust characteristics in selecting a set of hypotheses from the domain.

Corollary 2.4.1 For an arbitrary class C_a and a set of hypotheses $H_i s$ such as $H_i \cap C_a \neq \{\}$ and there is no conflicting instances (i.e. the one that are covered by more than one $H_i s$ whose decision classes are different), the following is true:

$$\begin{array}{rcll} \sum_i N_{H_i}^{pos} & = & \sum_i \max_{j=1...M} N_{H_i \cap C_j} & \textit{by Bayesian decision rule} \\ & \geq & \sum_i N_{H_i \cap C_a} & \textit{for any } C_{a=1...M}. \end{array} \tag{2.20}$$

Theorem 2.4.1 For a set of hypotheses $H_i s$ that is complete and non-conflicting in a whole problem domain, classification performance over the problem domain is greater than or equal to majority class prior probability, i.e., $P(C_d) = \max_i P(C_i)$.

Proof of Theorem 2.4.1

$$\begin{array}{rcll} \sum_i N^{pos}_{H_i} & = & \sum_i \max_{j=1...M} N_{H_i \cap C_j} & \textit{by Bayesian decision rule} \\ & \geq & \sum_i N_{H_i \cap C_a} & \textit{for any class } C_{a=1...M} \\ & \geq & \sum_i N_{C_a} & \textit{since } C_a \textit{ is fully covered by } H_i's \\ & \geq & N_{C_a} & \textit{for any } C_{a=1...M}. \end{array} \tag{2.21}$$

Hence, the following is true :

$$\sum_i N^{pos}_{H_i} \geq N_{C_d} \textit{ such that } P(C_d) = \max_{j=1...M} P(C_j). \tag{2.22}$$

Thus, classification performance of the set of rules (i.e., a hypothesis H_i and its decision class) is

$$\begin{array}{rcl} \textit{performance} & = & \dfrac{\sum_i N^{pos}_{H_i}}{\sum_i N_{H_i}} \\ & \geq & \dfrac{N_{C_d}}{\sum_i N_{H_i}} \textit{ such that } P(C_d) = \max_{j=1...M} P(C_j) \\ & \geq & \dfrac{N_{C_d}}{\sum_j N_{C_j}} \\ & \geq & P(C_d). \end{array}$$

Thus, any set of hypotheses that covers problem domain completely and does not include any conflicting hypotheses in it guarantees minimum performance of majority class prior probability, i.e., $P(C_d) = \max_j P(C_j)$. □

2.5 Summary

Comparing a neural network classifier with a Bayesian classifier provides probabilistic properties to the network and the rules extracted from the network. With the probabilistic view, it will be easier to understand the network and the rules. The comparison is summarized as:

1. A neural network directly estimates a posteriori probability, that is,

$$y_i(X) \approx P(C_i|X)$$

where $y_i(X)$ is the output value of i^{th} output node and C_i is the i^{th} class.

2. A neural network determines its decision class by Bayesian decision rule. That is, it assigns X to the C_i with maximal output value (i.e., a posteriori probability) given X.

3. The minimum performance of a Bayesian classifier is $P(C_d) = max_i P(C_i)$. With a simple setting of initial thresholds, a neural network satisfies the Bayesian minimum performance.

4. A posteriori probability $P(C_i|X)$ is interpreted as a probabilistic rule: *if* X, *then* C_i *with probability* $P(C_i|X)$.

5. The generality of a rule H is defined as $P(H) = \sum_i P(X_i)$ where X_i is covered by H.

6. A rule is called valid if it satisfies the following condition:

$$P(C_d|X_j) = \max_{i=1...M} P(C_i|X_j) \text{ for all } X_j's \text{ covered by } H.$$

7. A ruleset that is complete and non-conflicting in a problem domain satisfies the Bayesian minimum performance.

CHAPTER 3
DECOMPOSITIONAL APPROACHES

3.1 Introduction

Decompositional approaches to rule extraction from a trained neural network (i.e., a feedforward multilayered neural network trained with backpropagation algorithm) involves the following phases:

1. Intermediate rules are extracted at the level of individual units within the network.

2. The intermediate rules from each unit are aggregated to form the composite rule base for the neural network.

At each non-input unit of a trained network, n incoming connection weights and a threshold are given. Rule extraction at the unit finds a set of incoming attribute combinations that are valid (i.e., weight-sum of a combination is greater than the threshold) and maximally-general (i.e., size of each combination is as small as possible). Figure 3.1 gives an example to illustrate rule generation. For example, a combination (x_1, x_2) is a valid rule because its weight-sum (i.e., $3 + 4 = 7$) is always greater than the threshold (i.e., -1) regardless of the values of other incoming units. Another example is (not x_3) which is also valid. Gallant [10] describes a procedure to find a single rule to explain the conclusion reached by the neural network given a case. His method involves the ordering of attributes based on absolute weight values. However, he argues that the procedure work only if the network was very small; the number of implicitly encoded in-then rules can grow exponentially with the number of

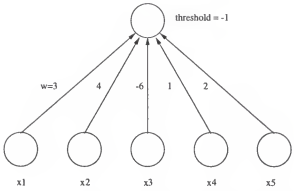


Figure 3.1. An individual rule with five incoming connection weights and a threshold.

incoming units [10]. There has been studies in generating a rule base with equivalent performance of the network [8, 7, 31, 25, 17]. Key issues in the efforts include:

1. completeness of the rule base,

2. quality of the generated rules and

3. computational complexity of the algorithm.

Fu's KT (KnowledgeTron) algorithm [6, 8, 7] searches valid rules for each hidden and output node of a trained neural network, and then rewrites those intermediate rules to exclude the symbols that do not correspond to predefined attributes or concepts. At each non input node, it searches the most simple but valid rule combinations among all possible combinations of incoming weight connections. In a search space tree, a root node is the most simple combination, i.e., no elements in it; combination nodes at depth one contains one element each and so on. A search begins from top to leaves of the tree, checking its validity. If it finds a valid rule combination, it stops generating children nodes and takes it as a rule. The key idea of the KT algorithm is separating positive and negative attributes in the search space to provide computationally efficient heuristics.

In addition to the heuristics, some parameters are introduced to control the complexity of the rule search procedure [9]. Some of the parameters include representative attributes, limitation of depth of the search tree, certainty factor threshold, rule simplification and merging similar rules, etc. The parameters help control the trade-off of quality of rules and searching complexity, but finding optimal values for the parameters is very difficult and mostly is accomplished trial and error. An automatic optimization procedure based on genetic algorithm was studied by Kim [12]. The GA-based optimization for setting parameter value can be justified because the parameter space is huge: they are continuous-valued and interact with each other.

Towell and Shavlik [31] developed the *MofN* algorithm. It explicitly searches for the rules of the form:

If M of the following N antecedents are true, then...

The idea underlying *MofN* is that individual antecedents do not have unique importance. Rather, groups of antecedents form equivalence classes in which each antecedent has the same importance as and is interchangeable with other members of the class. This idea allows the algorithm to consider groups of links without worrying about the particular links within the group. The steps of the *MofN* algorithm are following [31]:

1. With each non-input unit, form groups of similarly weighted links. This step is similar to Nowlan and Hinton's *soft weight sharing* [19].

2. Set link weights of all group members to the average of the group.

3. Eliminate any groups that do not significantly affect whether the unit will be active or inactive.

4. Holding all link weights constant, optimize biases of all non-input units using the backpropagation algorithm.

5. Form a single rule for each non-input unit. The rule consists of a threshold and weighted antecedents specified by the remaining links.

6. Where possible, simplify rules to eliminate superfluous weights and thresholds.

The algorithm makes two assumptions about trained networks. The first assumption is that the units are either maximally active or inactive. Thus, it is required for hidden units to be approximated as threshold units by assigning very high value to the gain parameter of sigmoid function, i.e., making it close to a hard-limiting function. The second assumption is that training does not significantly alter the meaning of units. Given that the *MofN* is essentially a rule refinement system, this may be true in general. However, in the case where the meaning of a unit does change during training, the comprehensibility of the extracted rules may be degraded significantly [1].

Other critiques have noted that the KBANN from which rules are to be extracted needs an initial rule set or weight clustering algorithm such as the soft weight sharing; special training is needed such as the one in step 4 above; the extracted rules use an intermediate term to represent each hidden unit. Because of these concerns the approach may not enable a sufficiently accurate description of the network to be extracted.

Setiono et al. [25, 17] proposed a rule-extraction approach, named RX, for a neural network. After a network is trained, the weight connections of the network are pruned. Classification rules then are extracted from the pruned network by the following algorithm:

1. Apply a clustering algorithm to find clusters of hidden node activation values.

2. Enumerate the discretized activation values and compute the network outputs. Generate rules that describe the network outputs in terms of the discretized hidden-unit activation values.

3. For each hidden unit, enumerate the input values that lead to them and generate a set of rules to describe the hidden units' discretized values in terms of the inputs.

4. Merge the two sets of rules obtained in the previous two steps to obtain rules that relate the inputs and outputs.

Tresp et al. [33] show how an initial rules can be used to prestructure a neural network of a set of multivariate Gaussian basis functions. They propose four different strategies for preserving the initial rule-based knowledge while still being able to refine the rule base during the training process. Their technique employs a probabilistic interpretation of the neural network architecture that allows the Gaussian basis functions to act as classifiers and that extracts and analyzes the refined rules.

Along with the decompositonal (open-box) approaches, there are studies of the black-box approaches too. One of the earliest published black-box approaches to rule extraction is that of Saito and Nakano [24, 23]. Their methods performs exhaustive search among the rule space spanned by attributes selected according to given instances. Rules extracted in this way may not cover unseen instances. This kind of black-box methods suffer difficulty of validity evaulation of rules. The RN (rule from networks) technique extracts rules from changes in the levels of the input and output units, and it can cope with continuous values. This method, however, generated a large number of rules even on a relatively simple problem domain.

The VIA (validity intervals analysis) technique developed by Thrun [29, 30] extracts rules that map inputs directly into outputs. The algorithm uses a generate-and-test procedure to extract symbolic rules from standard back-propagation neural

networks that have not been constructed specifically to facilitate rule extraction. The key idea is to attach intervals to the activation range of each unit (or a subset of all units, such as input and output units only), so that the network's activations must lie within these intervals. These intervals are called validity intervals. VIA checks whether such a set of intervals is consistent, i.e., whether there exists a set of network activations inside the validity intervals. It does this by iteratively refining the validity intervals, excluding activations that are provably in consistent with other intervals. It suffers from the drawback that it often will not find maximally-general rules because it makes the (sometimes incorrect) assumption that the activations of the hidden units are independent.

Craven [5, 4] proposed an approach called TREPAN to extract symbolic rules from neural networks. Craven frames the problem not as a search task, but instead as a supervised-learning task. The target concept is the function computed by the network; input features are simply the network's input features. The algorithm for extracting conjunctive rules from trained neural networks is outlined as follows:

```
/* initialize rules for each class */
for each class c
    R_c = {}
repeat
    e = EXAMPLES()
    c = classify(e)
    if e not covered by R_c then
       /* learn a new rule */
       r = conjunctive rule form from e
       for each antecedent r_i of r
           r' = r but with r_i dropped
```

```
            if SUBSET(c, r') = true then r = r'
        R_c = R_c + r
until stopping criterion met
```

The SUBSET module accepts a class label c and a rule r, and returns *true* if all instances covered by r are classified as members of class c by the network. For testing, Thrun's validity-interval analysis [29, 30] can be used or one of the open-box approaches can be used, where rules are extracted for each hidden and output unit individually. The EXAMPLES module is to provide an unlimited source of examples. Initially, EXAMPLES can return members of the network's training set. After the training set is exhausted, EXAMPLES finds examples by random sampling the instance space. One of the differences between this algorithm and search-based extraction approaches is that it explores the rule space from bottom to top.

3.2 More Issues in KnowledgeTron

3.2.1 KnowledgeTron in a Multi-Valued Domain

Given a hypothetical rule with a set of attributes, it is called valid when the weight sum exceeds a predefined threshold for all the instances covered by the rule. For binary attributes, Fu's KT algorithm [6, 8, 7] provides two methods: one for the set of positive attributes only (i.e., $H2_{KT}$) and another for the set of positive and negative attributes in its combination (i.e., $H3_{KT}$).

For multi-valued attributes, mutual exclusion relations between attributes should be considered. For example, three attributes A, B and C and each attribute have three values such as (a_1, a_2, a_3), (b_1, b_2, b_3) and (c_1, c_2, c_3), respectively. If $a_2 = 1$, then a_1 and a_3 should be 0. If $a_2 = 0$ (i.e., *not* a_2), then either a_1 or a_3 will be 1. For every attribute (i.e., A, B and C), the possible worst case weight is provided. For example, if there is a combination $(a_1, \text{not } b_2)$, then its possible lowest weight sum is

$w(a_1) + min(w(b_{j\neq 2})) + min(w(c_j))$. Given a hypothetical rule, it is valid if its lowest weight sum exceeds a certain threshold.

Another concern for the case of multi-valued attributes is rule simplification. During the rule-searching and rewriting process, the hypothetical rules need to be as simple as possible, eliminating redundant rules to keep the size of the ruleset small. Consider an attribute A involving n value elements. Thus, those n value elements are mutually exclusive in rules. Given a hypothetical rule, the number of positive attributes and the number of negatives are counted. The following three cases are checked:

1. If the number of positives is greater than 1, delete the rule. The case is "*if* a_1 *and* a_3," and it is wrong.

2. If the number of positives is one, we delete all negatives from the rule if any. The case is "*if* a_1 *and not* a_3 *and not* a_4," and the negatives are redundant.

3. If the number of positives is 0 and the number of negatives is $n-1$, add the positive value attribute not being included in the combination and remove all the negatives. The case is "*if not* a_2 *and not* a_3 *and* ... *not* a_n," and it can be simplified as *if* a_1.

3.2.2 Complexity Control Parameters

In generating rules from a neural network, Fu [8] develops several practical and useful heuristics for pruning the search space. In this section, additional strategies for controlling the complexity of this process are introduced.

As the size of a network increases, the computational complexity of the KT algorithm and the number of rules generated by the algorithm increase exponentially. Large numbers of those rules overlap each other and would degrade generalization of a KBCNN (Knowledge-Based Conceptual Neural Network) [6] that is built by the

rules. The rules need, therefore, refining so that only a few highly relevant rules could build KBCNNs, obtaining better generalization and smaller network size. Several techniques are integrated into KT algorithm to reduce the number of rules and refine the rules:

- When there are too many incoming connections to concept nodes, only highly weighted connections are considered.
- By setting the rule size small, the search space of the KT algorithm is reduced, and low relevant attributes are discarded.
- The CF(certainty factor) threshold (θ_{CF}) is used to discard the rules that have low certainty.
- The generated rules are refined by simplifying them using symbolic logic.
- The rules that have too many negative attributes are discarded.

If the number of input attributes is large, the complexity of the rule extraction process increases exponentially. In keeping the rule size small enough (e.g., 5), it is not necessary to consider all the input attributes. Thus, the small number of highly weighted connections (e.g., top 10 connections) is selected to form rules. This makes sense because input summation seems to be determined by a few highly weighted (highly relevant) connections, not by all the connections. But we cannot ignore the small valued connections also; so we normalized the weight values of the selected connection reflecting unselected connections.

$$\text{selected pos weight} = \text{selected pos weight} * \frac{\text{sum of all pos weights}}{\text{sum of selected pos weights}} \tag{3.1}$$

$$\text{selected neg weight} = \text{selected neg weight} * \frac{\text{sum of all neg weights}}{\text{sum of selected neg weights}} \tag{3.2}$$

When we select the highly weighted connections, we can create a ratio between the numbers of positive connections and negative connections. For example, we want to select the top 10 connections: 7 positives and 3 negatives. The ration is domain specific and determined empirically. Larger number of positives is preferred in order to introduce more positive attributes in a rule.

In the output layer, an output node reads connection weights from hidden nodes. If any hidden node does not have hidden rules, output nodes do not have to consider the connection from the hidden node, which reduces the search complexity.

Another parameter to control the number of rules generated is CF (certainty factor) values for each rules. Each rule has it's own CF, which is obtained in the following way:

$$CF = sigmoid(weighted\ sum - threshold). \tag{3.3}$$

If the CF is in $[0.5, 1.0]$, then this rule is a confirming rule and uses the CF. If the CF is in $[0.0, 0.5]$, the the rule is a disconfirming rule and convert the calculated CF of $[0.0, 0.5]$ into $[-1.0, -0.5]$ which is used as a CF of the rule. When we extract rules, we cannot consider all the possible rules because there are too many possible rules and there are many meaningless rules among them. Thus, we want only to select a few good rules. If the CF is high, the rule fine. The default CF threshold (θ_{CF}) used is 0.97, and only the rules whose CF are in $[0.97, 1.0]$ or $[-1.0, -0.97]$ are selected.

3.3 Applications: Network Refinement

The open-box approach assumes that each hidden and output node behaves like a concept formed by its incoming weighted connections that act as relationships between the connected concepts. Thus, it extracts rules from each concept node and combines those concept rules to find final rules. Consequently, this approach is used in many applications for the rule or domain theory refinement systems [6, 32, 31, 33]. In this section, we experiment with network refinement using the KT algorithm. An

initial neural network is trained on data, interpreted by the KT algorithm, and the extracted rules are used to re-structure (i.e., refine) the network. Performances of the original network and the refined network are evaluated and compared. Part of the work presented here is also described in [13, 9].

First, a rule-based neural network is described and its generalization capability is compared with a standard neural network. Then, network refinement is sought using the KT algorithm and rule-based neural network techniques.

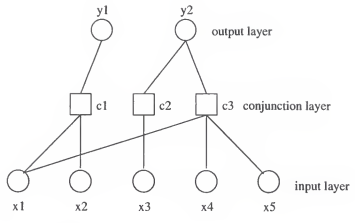
3.3.1 Rule-Based Neural Networks

There have been studies on knowledge-based neural networks that use domain knowledge to determine the initial structure of the neural network. There are different models of the knowledge-based neural networks: rule-based, decision-tree based and semantic-constraint based. Examples of the rule-based neural networks are KBCNN [6] and KBANN [32], which adopt different weight initialization schemes depending on the type of activation function chosen.

In the knowledge-based conceptual neural network (KBCNN) model, a set of production rules can be mapped into a neural network structure. Where attributes or variables are assigned to input units, target concepts or final hypotheses are assigned to output units; intermediate concepts or hypotheses are assigned to the hidden units. The rules determine the connections of the units and the weights of the connections [6].

A rule with a conjunction of attributes is assigned to a conjunction unit which combines the conditions in the premise of the rule. Thus, each conjunction unit represents a rule and is assigned to an output node according to the consequence of the rule. Figure 3.2 illustrates the structure of KBCNN over a set of rules.

The initial weights of the connections can be determined in the following way. Suppose that a rule's premise involves p positive attributes and q negated attributes.



Rules: if x1 and x2, then y1
if x1 and x4 and x5, then y2
if x3, then y2

Figure 3.2. Structure of KBCNN given a set of rules.

The initial threshold of the corresponding conjunction unit is set to about 0.2, and the initial weight of each input connection from positive attributes is set to around $1/p$ (or $-1/q$ for the connection from negated attributes). The weights from conjunction nodes to output or disjunction nodes are set to belief values attached to the rules of the conjunction nodes or set to strong values (e.g. 0.5) [6].

Given a training data set, it is important to decide the optimal complexity of a network for the domain data in terms of the hidden layer size and the number of connections. In the extreme case when the number of hidden nodes is the same as the number of training examples, each hidden pattern in a hidden layer corresponds to each input pattern and thus no generalization occurs. If the complexity is too low, the network can not accommodate all the hidden patterns that the input patterns represent. There are two approaches to find the right complexity of the network. Top-down methods reduce the hidden layer size to an optimal size, as in the case of skeletonization pruning [18], weight decay [15], gain decay [16], etc. Bottom-up

methods increase the complexity up to the optimum from minimum, as in the case of cascade correlation and Rule-Based Neural Network (RBNN) architecture.

The complexity of the RBNN is determined by the complexity of the initial rules. If it is not trained, the network constructed by the rules simulates a rule-based system. The network, however, can learn more than the initial rules by training with examples. Thus, learning requires more complexity. The RBNN suffers much less from the over-fitting problem which is caused by overtraining and an unbalanced training data set. We showed that adding a few extra hidden nodes and connections improves the generalization of the RBNN [13].

The *IRIS* rules shown below are used for experiments. It is shown that the RBNN is more tolerant to overfitting problems and thus has better generalization capability than standard networks.

```
SIX RULES:
R1: If petal-length <= 2.7,
    Then it is setosa.
R2: If 2.7 < petal-length <= 5.0, and
       0.7 < petal-width <= 1.6,
    Then it is versicolor.
R3: If petal-length > 5.0,
    Then it is viginica.
R4: If septal-width <= 2.8, and
       petal-width > 1.6,
    Then it is viginica.
R5: If 2.8 < septal-width <= 3.1, and
       petal-width > 1.6,
    Then it is viginica.
R6: If septal-length > 3.1, and
       2.7 < petal-length <= 5.0,
    Then it is versicolor.
```

In our experiment, generalization was measured by

```
Gen = (number of correct classifications on test data set)/
      (number of total examples on test data set).
```

Given the complexity of the network, if the size of the training data set is too small, then the network can overfit the training examples and degrade generalization. In the case of a standard network with 6 hidden units and full connections between layers, generalization was decreased from 96.22% to 90.49% when the size of training data set was reduced from 50 to 10 instances. However, in the case of the RBNN, generalization was decreased from 93.56% to 90.67%, which can be improved up to 91.60% by adding few extra hidden units that are fully connected to other layers with random initial weights. Figure 3.3 shows the generalizations of a regular network, an RBNN without extra hidden units, and RBNNs with 3 and 5 extra hidden units. The RBNNs have better generalization when a small numbers of training examples are available. An RBNN with 3 extra hidden units has better generalization than one with 5 extra hidden units. The result of Figure 3.3 explains the relationship between training data set and an appropriate complexity of the network.

Secondly, we investigated how overtraining affects generalization. We used a training data set of 30 instances with one of them as an ambiguous or corrupt example. When overtrained, the regular network seemed to learn even the corrupt one. As a result, generalization is degraded (see Figure 3.3(a)). However, the RBNN does not learn the corrupt rule, thus it does not lose generalization (see Figure 3.3(b)). This is so because of the rule-based connections which constrain the direction of weight vectors. The RBNN has the ability to exclude corrupt examples in a training data set.

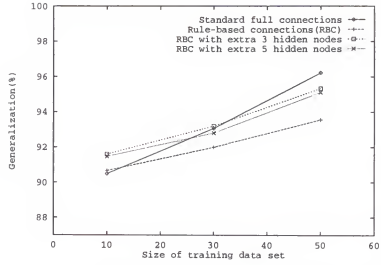
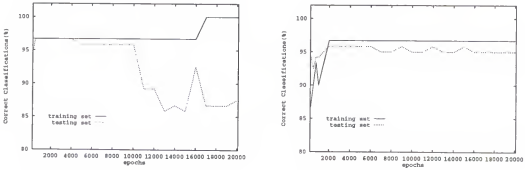


Figure 3.3. In the case of small size of training data set, RBCNs have better generalization than a tested standard multi-layered network.



(a) Standard fully connected network

(b) Rule-based connectionist network

Figure 3.3. Overtraining degrades generalization capability:(a). But RBCN is not influenced by the overtraining:(b).

3.3.2 Network Refinement

Performance comparison between KBCNNs and standard 3-layered feed-forward networks is accomplished in terms of generalization and network size. The standard networks are fully connected between layers, while KBCNNs have rule-based connections. The promoter domain has 106 instances. Each instance consists of a DNA nucleotide string of four base types: A(adenine), G(guanine), C(cytosine), and T(thymine). Each instance string involves 57 sequential nucleotides. An instance is a positive instance if the promoter region is present in the string; else it is a negative instance. There are 53 positive and 53 negative instances. In the Hepatitis domain, there are 155 instances, each described by 19 features: six continuous features and thirteen nominal features. The Fisher's iris data set contains 3 classes with 50 instances each, and each class refers to a type of flower, the iris. One class is linearly separable from the other two which are not linearly separable from each other. Each instance in the data set is described by four continuous features.

The process of experiments is shown in Figure 3.4, and the performance comparison is done between the trained standard network and the trained KBCNN. The generalization was evaluated by cross-validation, in which each domain set is divided into two subsets and a network is trained by one subset and tested by the other. Table 3.1 shows each domain's network configurations which specify the number of units in each layer in the form of (*input layer*)-(*hidden layer*)-(*out layer*) for standard networks and (*input layer*)-(*first hidden layer*)-(*second hidden layer*)-(*third hidden layer*)-(*output layer*) for KBCNNs. The KBCNNs are constructed with extracted rules as seen in Figure 3.4. Table 3.2 shows a set of production rules that are extracted from a standard network trained with promoter data set1. The performance comparison is shown in Table 3.3. The result of the promoter domain is quite promising because the KBCNN improves generalization with 20 times smaller number of

Figure 3.4. Experiment Procedure.

		Network configuration	
		KBCNN	Standard NN
Promoter Data	set1	228-6-5-5-1	228-14-1
(0 potential node)	set2	228-13-7-18-1	
Promoter Data	set1	228-6-6-5-1	228-14-1
(1 potential node)	set2	228-13-8-18-1	
Hepatitis Data	set1	34-39-18-9-2	34-20-2
	set2	34-30-17-11-2	
Iris Data	set1	12-21-3-3-3	12-6-3
	set2	12-16-4-8-3	

Table 3.1. Network configurations.

connections. When a potential node is added to the KBCNN, generalization is improved even more, even though network size increases. In the hepatitis domain, the KBCNN has better generalization with only 37.7% of the size of standard network. The iris domain has a small number of input attributes, so the network size is small. The same generalization was obtained with a smaller number of connections.

An important parameter in the rule extraction process is the Certainty Factor (CF) threshold (θ_{CM}). The CF threshold handles the certainty of rules by discarding the rules that have lower certainty factors than the threshold. This refines rulesets and reduces network size, and thus improves generalization of the KBCNN that is built up by the rules. As a higher CF threshold (θ_{CF}) is applied, only high certainty rules are extracted, and thus the number of connections is decreased and generalization is increased (Figure 3.5 and Figure 3.6).

R1	If @–12 "ā", @–31 "cxgxt̄"	Then $\overline{\text{KTNC}-0}$
R2	If @+3 "g", @–12 "a", @–18 "t", @–31 "axḡgtt"	Then KTNC–0
R3	If @–2 "t", @–11 "g", @–18 "t̄", @–31 "ctgḡat̄", @–45 "g"	Then KTNC–2
R4	If @–9 "t", @–12 "a", @–18 "t", @–31 "axḡgtt"	Then KTNC–5
R5	If @–9 "g", @–12 "c", @–31 "ctgḡaa"	Then KTNC–6
R6	If @–2 "t", @–12 "c", @–31 "cxgḡaa", @–43 "c"	Then KTNC–8
R7	If KTNC–0, KTNC–5, $\overline{\text{KTNC}-6}$	Then promoter
R8	If KTNC–6, KTNC–8	Then nonpromoter
R9	If KTNC–6, KTNC–2	Then nonpromoter
R10	If KTNC–8, KTNC–2, $\overline{\text{KTNC}-0}$	Then nonpromoter
R11	If KTNC–8, KTNC–2, $\overline{\text{KTNC}-5}$	Then nonpromoter

Table 3.2. The extracted rules from the trained standard neural network on the Promoter data.

		KBCNN			Standard NN		
		set1	set2	avg	set1	set2	avg
Promoter Data	Generalization(%)	84.91	86.79	85.85	81.11	84.91	83.02
(0 potential node)	Connections	96	217	156.5	3221		
Promoter Data	Generalization(%)	92.45	84.91	88.68	81.11	84.91	83.02
(1 potential node)	Connections	320	449	386.5	3221		
Hepatitis Data	Generalization(%)	96.10	96.15	96.13	94.81	93.59	94.19
	Connections	272	288	280	742		
Iris Data	Generalization(%)	98.67	97.33	98.00	98.67	97.33	98.00
	Connections	136	131	133.5	234		

Table 3.3. Generalization and the number of connections of KBCNNs and standard neural networks on three data domains.

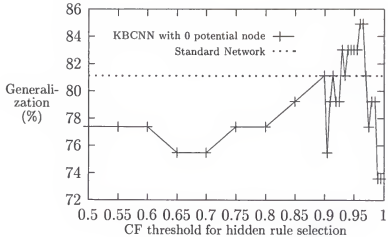


Figure 3.5. Generalization with different CF thresholds for hidden rule selection (Promoter data). The CF threshold for output rules was set to 0.6.

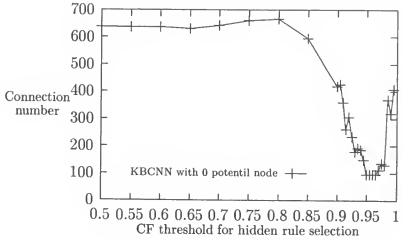


Figure 3.6. The number of connections in KBCNNs with different CF thresholds for hidden rule selection (Promoter data). The number of connections of the standard neural network is 3221.

3.4 Discussions

One main issue in open-box approaches is the search complexity. The open-box algorithms extract rules at the level of individual units of a trained neural network. Thus, complexity of the rule-search at each unit increases exponentially with the number of incoming connections. And the total complexity for the network increases with the number of hidden and output units. In addition to the rule searching complexity, rule rewriting complexity is also problematic.

Several approaches have been studied to reduce the searching complexity of open-box methods. Fu [7] introduced the parameter *bound* that limits the size of rules to be searched for his KT rule extraction procedure. Fu and Kim [9] proposed more parameters that controls the complexity such as representative attributes, ration of positive and negative attributes, certainty factor threshold and others. The parameter setting approach, however, also has a problem: the optimal set of parameters should be determined for optimal performance. Kim [12] presented a GA approach to find an optimal set of parameters for the best set of rules to be extracted.

Some other approaches include the soft-weight-sharing used in *MofN* algorithm [31]. It clusters the weights into equivalence classes. This approach, however, requires a special training technique and is not sufficiently accurate description of the network.

The proposed "ordered-attribute" search algorithm described in the next section reduces the search space dramatically. At each node, it finds b rules that are valid and maximally general after very few hypotheses are checked.

Another problem that the open-box approaches might have is the discontinuity problem defined here. The rule extraction procedure at each unit assumes boolean inputs and boolean activations. For hidden units, inputs are discrete binary values. However, activation is not boolean, but continuous in the range of 0 and 1. For output nodes, inputs are activations from the hidden units which are not boolean

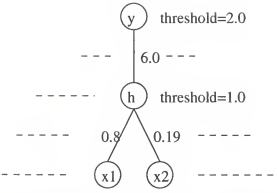


Figure 3.7. This example illustrates discontinuity problem of some individual unit-based rule extraction methods.

values. Thus, the approaches use a kind of hard-limiting activation functions for hidden units. This can be obtained by setting high gain values in sigmoid functions of the hidden units. Namely, it is discretization of activation values of the hidden units into two discrete values, i.e., 0 and 1 (e.g., KT and MofN), or more than two values by clustering (e.g., RX [25, 17]).

The assumption requires that the hidden units of the network be approximated by threshold units as follows:

$$h_j = \begin{cases} 1, & \text{if } \sum w_{ji} x_i > \theta, \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

Thus, the extracted rules may not provide an accurate representation of the network. A simple example is illustrated in Figure 3.7. Suppose we have an input vector in which x_1 and x_2 are 1's and the other elements are 0's. Then, obviously the rule "*if* $x_1 = 1$ *and* $x_2 = 1$, *then* y" is true or subsumed to any rule that is true. However, unit-based rule extraction methods make the rule false. At unit h, the combination of "$x_1 = 1$ *and* $x_2 = 1$" is rejected because the weighted sum does not exceed the threshold of h. Thus, the rule "*if* $x_1 = 1$ *and* $x_2 = 1$, *then* y" is not true if the unit-based rule extraction methods are used.

CHAPTER 4
ORDERED-ATTRIBUTE SEARCH

4.1 Introduction

In the previous chapter, we proposed several heuristics to reduce the search space without degrading the quality of rules to be found. In this section, we propose a different search algorithm called the *ordered-attribute search* (OAS) algorithm. The OAS algorithm involves the following three procedures:

1. **Attribute Contribution Scoring:** When an attribute is added to a combination(i.e., a candidate rule), the weight sum of the combination is increased (or decreased). The contribution score of each attribute is defined by the amount of increase (or decrease) when the attribute is added to a candidate rule.

2. **Attribute Sorting:** Attributes are sorted in descending order according to their contribution scores.

3. **Rule Searching:** With the attributes sorted by their contribution scores, valid and maximally general rules are to be searched.

This algorithm is very efficient in terms of computational complexity: it costs $O(n \log n)$ to search for a maximally general rule that is valid. Note that traditional algorithms costs $O(2^n)$ where n is the number of attributes. Complexity for searching for b rules in the OAS is $O(C_l^n)$ while other traditional algorithms costs $O(\sum_{l=1}^{n} C_l^n) = O(2^n)$.

4.2 Attribute Contribution

Given a set of incoming weights $(w_1, \ldots, w_i, \ldots, w_n)$ to a node, it is simple to find an instance $(x_1, \ldots, x_i, \ldots, x_n)$ in which its weight sum, $\sum_{i=1}^{n} w_i x_i$ is higher or lower

than any other $2^n - 1$ instances.

$$\text{sum}_{highest} = \sum_{i=1}^{n} w_i x_i \text{ st. } \begin{cases} x_i = 1 & \text{if } w_i > 0 \\ x_i = 0 & \text{if } w_i < 0 \end{cases} \tag{4.1}$$

$$\text{sum}_{lowest} = \sum_{i=1}^{n} w_i x_i \text{ st. } \begin{cases} x_i = 1 & \text{if } w_i < 0 \\ x_i = 0 & \text{if } w_i > 0 \end{cases} \tag{4.2}$$

Since the sum_{lowest} is the lowest among all possible instances, it is also a weight sum of a *default rule* (length 0) that covers all the possible instances. Increasing the length of a default rule by adding attributes increases the weight sum of the rule.

For binary attribute cases, the contribution score $C(x_i)$ of an attribute x_i in terms of the weight sum is defined as follows:

$$C(x_i) = |w_i|. \tag{4.3}$$

If $w_i > 0$, a positive attribute (i.e., $x_i = 1$ or "*if* x_i") increases the sum_{lowest} by $|w_i|$. If $w_i < 0$, a negative attribute (i.e., $x_i = 0$ or "*if not* x_i") increases the sum_{lowest} by $|w_i|$.

For the case of multi-valued attributes, $C(x_i)$ is calculated a little differently. For example, an attribute A has three values a_1, a_2 and a_3. Each value element of A behaves like a binary attribute. Those binary attributes of A are, however, mutually exclusive. Therefore, "*if* a_1" means "$a_1 = 1$ *and* $a_2 = 0$ *and* $a_3 = 0$." Likewise, "*if not* a_1" means "$a_2 = 1$ *or* $a_3 = 1$." The lowest weight sum is

$$\text{sum}_{lowest} = \sum_{i=1}^{n} min(w^i) \tag{4.4}$$

when n is the number of attributes and $min(w^i)$ is the minimum weight value. The contribution score $C(x_k^i)$ of the k^{th} value element of the i^{th} attribute is defined as follows:

$$C(x_k^i) = \begin{cases} -min(w^i) + w_k^i & , \text{ if } w_k^i \geq 0 \\ -min(w^i) + min(w_{j \neq k}^i) & , \text{ if } w_k^i < 0. \end{cases} \tag{4.5}$$

Thus the weight sum is increased by the amount of $C(x_k^i)$ if an attribute condition $x_k^i = 1$ (for $w_k^i \geq 0$) or $x_k^i = 0$ (for $w_k^i < 0$) is added to a default rule. For example,

consider a multi-valued attribute A with the three values whose connection weights are $(3, -5, -2)$. Then, $min(A) = min(3, -5, -2) = -5$. Thus, the contribution score of the 1^{st} value element is $-(-5) + 3 = 8$. The score of the 2^{nd} value element is $-(-5) - 2 = 3$ and that of the 3^{rd} is $-(-5) + 5 = 0$.

Lemma 4.2.1 *The contribution score of an attribute is not negative.*

Proof of Lemma 4.2.1 *For a binary attribute x_i, $C(x_i)$ is always greater than or equal to 0 by the definition in 4.3. For a multi-valued attribute x_i, contribution score of its element, $C(x_k^i)$, is always non-negative because:*

1. *if $min(w^i) \geq 0$, $w_k^i - min(w^i) \geq 0$ since $w_k^i \geq min(w^i)$. Thus, $C(x_k^i)$ is non-negative by the definition in 4.5.*

2. *if $min(w^i) < 0$, $-min(w^i) \geq w_j^i$ for any j. Thus, $-min(w^i) + min(w_j^i)$ is non-negative.*

Therefore the contribution score is always non-negative. □

Definition 4.2.1 *Suppose that l is a length of combinations. The $csum(a_1, a_2, \ldots, a_l)$ is defined as a sum of contribution scores of l attributes in the parentheses. The l attributes are in descending order according to their contribution scores.*

Lemma 4.2.2 *Given a set of weights W and a rule R (i.e., candidate combination), the csum of R is proportional to the lowest weight sum out of all the instances that belong to the rule.*

Proof of Lemma 4.2.2 *By definitions of the csum and the contribution score, $csum(R) + sum_{lowest}(W) =$ the lowest weight sum of R. Since the $sum_{lowest}(W)$ is constant for any combination defined on W, the $csum(R)$ is proportional to the lowest weight sum.*

□

Proposition 4.2.1 *Given a combination R defined on a set of weights W, the R is not valid if* $(csum(R) + sum_{lowest}(W))$ *does not exceed a validity threshold.*

4.3 Attribute Sort

After the contribution score of each attribute is calculated, the attributes are sorted in descending order by their contribution scores. The complexity of well-known sorting algorithms (i.e., quick sort) is $O(n \log n)$ where n is the number of attributes. Let us define a set of n ordered attributes as follows:

$$a_1, a_2, a_3, \ldots a_n. \tag{4.6}$$

Given a set of ordered attributes, there are several ways to list the combinations of length l. Note that the number of combinations of length l is C_l^n. In this paper, we use the generic algorithm that lists the combinations in the following order:

$$(a_1, a_2, \ldots a_l), \ldots (a_1, a_2, \ldots a_n), \ldots (a_1, a_3, \ldots a_{l+1}), \ldots (a_{n-l+1}, \ldots a_n). \tag{4.7}$$

For example, let us say the n is 10 and the l is 3. Then the 120 (i.e. C_3^{10}) combinations are ordered in the following order:

$$(a_1, a_2, a_{k=3..10}), (a_1, a_{j=3..9}, a_{k=(j+1)..10}), (a_{i=2..8}, a_{j=(i+1)..9}, a_{k=(j+1)..10}).$$

An OAS Search tree is composed of $n + 1$ depths, starting from a root node of length 0 and ending to leaf nodes of length n. At each depth l (i.e., $1 \leq l \leq n$), the number of combinations is C_l^n and they are listed from the left to the right as defined at (4.7). The total number of combinations in the OAS search tree is $\sum_{k=0}^{n} C_k^n = 2^n$.

Corollary 4.3.1 *In the list of combinations of length l, the first combination (i.e., the left-most one) holds the highest csum. The last one in the list holds the lowest csum.*

Proposition 4.3.1 *By the definition of csum in 4.2.2 and combination listing in 4.7, the following inequalities are true:*

$$
\begin{array}{lll}
1. & csum(a_{s_1}, a_{s_2}, \ldots, a_{s_l}) & > csum(a_{s_1}, \ldots, a_{s_{l-1}}, a_{i_l > s_l}) \\
2. & csum(a_{s_1}, \ldots, a_{s_j}, \ldots, a_{s_l}) & > csum(a_{s_1}, \ldots, a_{s_{j-1}}, a_{i_j > s_j}, a_{i_{j+1} \geq max(s_{j+1}, i_j+1)}, \\
 & & \quad \ldots, a_{i_l \geq max(s_l, i_{l-1}+1)}) \\
3. & csum(a_{s_1}, a_{s_2}, \ldots, a_{s_l}) & > csum(a_{i_1 > s_1}, a_{i_2 \geq max(s_2, i_1+1)}, \\
 & & \quad \ldots, a_{i_l \geq max(s_l, i_{l-1}+1)}).
\end{array}
\tag{4.8}
$$

Definition 4.3.1 *Given a combination $R = (a_{s_1}, a_{s_2}, \ldots, a_{s_l})$, the set of all combinations with smaller csums, as defined by the three inequalities in Proposition 4.3.1, is defined as the V set of the combination R.*

4.4 Rule Search

Due to Corollary 4.3.1, it is straightforward to find the smallest number of attributes whose weight sum exceeds any defined threshold. Starting from the depth 1 at an OAS search tree, only the first node (i.e., combination) at each depth is tested for validation. If the node is not valid, the rest at the depth is not valid (by Corollary 4.3.1). Table 4.1 illustrates an algorithm for the procedure and its computational complexity is $O(n)$. This approach tests just d combinations before the smallest size is determined.

Lemma 4.4.1 *Let d be the depth found at the algorithm in Table 4.1. All the combination nodes from depth 0 to depth (d-1) are not valid.*

Proof of Lemma 4.4.1 *By the algorithm in Table 4.1, all of the first (i.e., left-most) combination from depth 0 to $(d-1)$ are not valid. If the first combination at depth l is not valid, then all combinations at the depth l are also not valid by Corollary 4.3.1. Thus, all combinations from depth 0 to $(d-1)$ are not valid.* □

attributes $a_1, a_2, \ldots, a_n$ are ordered in descending order.
01. sum=0
02. d=0
03. while(sum< θ) {
04. d++
05. sum=sum+a_d
06. }
07. return($a_1, \ldots, a_d$)

Table 4.1. An algorithm for finding a combination which is valid and has the lowest length d.

Lemma 4.4.1 makes it possible to reduce the search space by $\sum_{k=0}^{d-1} C_k^n - d$.

We want to find a set of b rules that are valid, such that (1) they are as general as possible and (2) their weight sums are as high as possible. Once the smallest size d is obtained, total C_d^n combinations at the depth d are subject to searching for the b combinations which have higher weight sums than other $(C_d^n - b)$ combinations. By the three inequalities in proposition 4.3.1, the search space at depth d is reduced dramatically.

Lemma 4.4.2 *Given a combination at depth d, if the combination is not valid, combinations of its* V set *are not valid either.*

Proof of Lemma 4.4.2 *If a combination R is not valid, its* $(csum(R) + sum_{lowest}(W))$ *is less than a validity threshold (by Proposition 4.2.1). Since the* $sum_{lowest}(W)$ *is constant for every combination and csum(R) is greater than csum of any combination in the V set, every combination in the V set is not valid.* □

Validity testing begins with the first combination at depth d. As each combination in the list is tested, valid one is stored in a rule candidate set. If it is not valid, its *V set* is eliminated from the search space instantly. For example, let the smallest size l be 3 and the number of attributes n be 10. Then the number of possible combinations

is $C_3^{10} = 120$. Consider a combination $R = (a_1, a_3, a_6)$. By the inequality 1 in proposition 4.3.1, weight sum of R is greater than the ones of $(a_1, a_3, a_7) \ldots (a_1, a_3, a_{10})$. It eliminates $\sum_{k=6+1}^{n} 1 = (10 - 6) = 4$ combinations in the search space. By the inequality 2, weight sum of R is greater than the ones of $(a_1, a_{j>3}, a_{k \geq max(6,j+1)})$. It eliminates $\sum_{j=3+1}^{n-1} \sum_{k=max(6,j+1)}^{n} 1 = 20$ combinations. By the inequality 3, weight sum of R is greater than the ones of $(a_{i>1}, a_{j \geq max(3,i+1)}, a_{k \geq max(6,j+1)})$. It eliminates $\sum_{j=1+1}^{n-2} \sum_{j=max(3,i+1)}^{n-1} \sum_{k=max(6,j+1)}^{n} 1 = 80$ combinations. Weight sum of the R is greater than weight sums of 104 other combinations. Hence, given the combination R, lots of combinations can be excluded from the search space. This heuristic reduces the search space dramatically.

If the number of rules found at the depth d is less than b, the search goes to the next depth and goes on down to the leaf depth until b valid rules are found. At depth l (i.e., $l > d$), another space reduction algorithm is used in addition to the space reduction by the three inequalities. At depth l, if a combination is valid, its child combinations at depth $l + 1$ are eliminated from the search space because they are subsumed to the parent.

Definition 4.4.1 *Given a combination $(a_{s_1}, a_{s_2}, \ldots a_{s_l})$ at depth l, its child combinations at depth $l + 1$ are $(a_{s_1}, a_{s_2}, \ldots a_{s_l}, a_{(s_l+1)\ldots n})$.*

Lemma 4.4.3 *If a combination at depth l is valid, its child combinations at depth $l+1$ are also valid.*

Proof of Lemma 4.4.3 *Since the combination is valid, its csum plus sum_{lowest} is greater than a validity threshold by proposition 4.2.1. The csum of its child node is greater than or equal to the one of the parent because a new $C(a_{k,k=(s_l+1)\ldots n})$ is added to the parent's csum and the $C(a_k)$ is always non-negative (by lemma 4.2.1). Since sum_{lowest} is constant for every combination in the search tree and the new csum is*

greater than or equal to the parent's, the child's csum plus sum_{lowest} *is greater than a validity threshold. Therefore it is also valid.* □

Thus, at depth $l > d$, two space reduction algorithms are used: one by Lemma 4.4.2 and the other by Lemma 4.4.3. In addition, Lemma 4.4.1 provides another space reduction algorithm as described before. Those three space reduction algorithms are provided by the OAS framework. A brief algorithm is illustrated in Table 4.2.

```
01.   find the smallest size, l, of combination
      whose weight sum is greater than a threshold, θ.
02.   index_b = 0
03.   while(index_b < b) {
04.      find the highest weight sum combination of size l .
05.      while(SUM > θ and index_b < b) {
06.         put the combination to a ruleset.
07.         index_b++
08.         find the next highest weight sum combination of size l.
09.         }
10.      if(SUM ≤ θ and index_b < b)
11.         l++
12.      }
```

Table 4.2. The Ordered Attribute Search Algorithm

Suppose that the 120 (i.e., C_3^{10}) combinations are ordered in the following order:

$$(a_1, a_2, a_{k=3..10}), (a_1, a_{j=3..9}, a_{k=(j+1)..10}), (a_{i=2..8}, a_{j=(i+1)..9}, a_{k=(j+1)..10}).$$

The OAS searches through the C_3^{10} combinations from (a_1, a_2, a_3) to (a_8, a_9, a_{10}) in the order defined above to find the b combinations which are valid and have weight sums greater than those of others. During the searching process, if the current combination is not valid, the combinations with smaller weight sums can be eliminated from the search space. Figure 4.1 illustrates the number of combination that can be eliminated when current combination (i.e., combination id) is not valid.

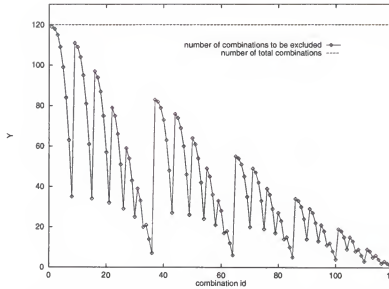


Figure 4.1. Number of combinations to be eliminated from search space when current combination is not valid.

The OAS reduces search space even when it tries to find $b << C_3^{10}$ valid combinations. To find 1, 2 or 3 best rules out of C_3^{10} combinations, the OAS scans only 1, 2 or 5 combinations respectively, instead of scanning of $C_3^{10} = 120$ combinations. Figure 4.2 illustrates the number of combinations (in a $\log_2$ scale) to be scanned to find 1, 2 or 3 best rules out of C_l^{10} combinations with $l = 1 \ldots 9$.

The worst case comes when it tries to find $b = C_3^{10}$ combinations from C_3^{10} combinations in which all are valid. However, usually b is much less than C_3^{10}, and not all of the combinations are valid.

In conclusion, the OAS algorithm is complete and generates sound rules with very effective computational complexity.

Theorem 4.4.1 *The rules generated by the OAS algorithm are sound.*

The combinations stored in the rule set are valid by the OAS algorithm.

Theorem 4.4.2 *Let n be the number of valid and not-subsumed rules in the OAS search tree. If $b \geq n$, then the OAS can generate a complete set of rules by an*

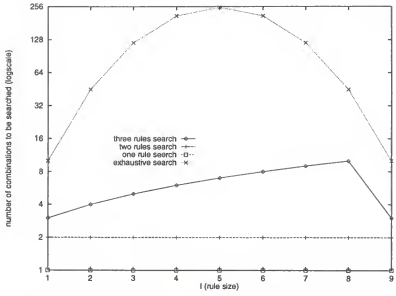


Figure 4.2. Number of combinations to be scanned (log scaled) to find 1, 2 or 3 best rules.

appropriate parameter values. That is, all valid rules are either in the generated rule set or subsumed by some other rule in the set.

From the OAS search tree, only the following types of combinations are eliminated from the search tree:

1. invalid combinations (Lemma 4.4.1 and 4.4.2) and

2. valid combinations which are subsumed by valid parent combinations (Lemma 4.4.3).

All other combinations at the tree are subject to a validity test for rule candidacy.

4.5 Complexity Comparison with Related Algorithms

Given n binary attributes, there are, in total, 2^n different instances in the input domain and 3^n different rules in the rule space. The number of combinations of length l is C_l^n. For each combination, 2^l binary representations are possible. Hence there are $2^l \cdot C_l^n$ possible rules of length l. The number of total possible rules, $\sum_{l=0}^{n} 2^l \cdot C_l^n$, is obtained by binomial expansion:

$$(x+y)^n = \sum_{l=0}^{n} C_l^n \cdot x^l y^{n-l}. \tag{4.9}$$

The $\sum_{l=0}^{n} 2^l C_l^n$ occurs when $x = 2$ and $y = 1$:

$$3^n = \sum_{l=0}^{n} 2^l C_l^n. \tag{4.10}$$

Given a combination of length l, however, we are interested in its validity only. Its validity is determined by the binary encoding with the lowest weight sum: if its weight sum exceeds a validity threshold, all 2^l encodings of the combination have weight sums greater than the threshold and thus are all valid. Therefore, given a combination, only its lowest weight sum is tested for validity and the lowest weight sum, sum_{lowest} is defined in Equation 4.2. This reduces rule search space to:

$$2^n = \sum_{l=0}^{n} C_l^n. \tag{4.11}$$

This provides the following corollaries which will be used in complexity comparisons later in this dissertation.

Corollary 4.5.1 *The number of possible rules of length l is loosely bounded by 2^n. That is,*

$$C_l^n < 2^n$$

Corollary 4.5.2 *The following equality is true*

$$2^{n-1} = \sum_{l=0}^{\frac{n}{2}} C_l^n$$

where n is an even number.

Corollary 4.5.3 *The number of combinations of length $\frac{n}{2}$ is loosely bounded by 2^{n-1}. That is,*

$$C_{\frac{n}{2}}^n < 2^{n-1}$$

In the problem of rule extraction from a neuron unit which involves n incoming binary attributes, we want to find the b rules which are *valid* and maximally general (i.e., the rules having *lower length* than other rules). Even though there are 2^n possible rules, only $C^n_{\lfloor n/2 \rfloor}$ unique rules are possible in maximum because higher length rules are subsumed to lower length rule. Hence the following is true:

$$1 \leq b \leq C^n_{\lfloor \frac{n}{2} \rfloor}.$$

Given a rule, validity evaluation costs $O(1)$ and the complexity of finding the b best rules depends on the complexity of the search algorithm.

In this section, we categorize the search algorithms as follows:

1. Exhaustive search (EXHS) searches through all combinations to select b best rules.

2. Tree-based search (TRS) searches through all the combinations stored in a tree structure. The tree structure provides some heuristics for search space reduction. KT algorithm is an improved version that utilizes efficient heuristics.

3. Ordered Attribute search (OAS) uses tree structure, but combinations are stored according to their contribution scores. This structure provides many efficient algorithms and heuristics that reduce the search space dramatically in most cases.

Exhaustive search (EXHS) costs $O(\sum_{l=0}^{n} C^n_l) = O(2^n)$ for any b in the worst case:

$$O(EXHS) = O(\sum_{l=0}^{n} C^n_l) = O(2^n) \text{ for any } b. \tag{4.12}$$

Tree search (TRS) uses a tree structure where each node is one of 2^l combinations. The TRS search begins with a root node (i.e., the lowest length combination) to a leaf node (i.e., the largest length combination). The tree search makes use of the following property:

if a combination of a node is a valid rule, combinations of its descendant nodes are eliminated from the search tree space.

This property reduces search space. For $b = 1$, the worst case occurs when the rule is a leaf node combination of length n. The worst case for any $b > 1$ occurs when the b rules are length l combinations such as $C_{l+1}^n < b \leq C_l^n$ where $\lceil \frac{n}{2} \rceil \leq l < n$. Its worst-case search cost for any b is :

$$O(\sum_{l=0}^{\lceil \frac{n}{2} \rceil} C_l^n) \leq O(TRS) \leq O(\sum_{l=0}^{n} C_l^n). \tag{4.13}$$

The KT algorithm is an improved tree search algorithm which reduces search space and memory space heuristically. The KT algorithm reduces the search space by dividing the n incoming attributes into p positive and q negative attributes (st. $p+q = n$). The basic idea of the separation derives from the fact that the complexity of an exponential space is greater than the sum of complexities of its sub-spaces. That is,

$$x^l \geq (a \cdot x)^l + (b \cdot x)^l \tag{4.14}$$

where $l > 1, a < 1, b < 1$ and $a + b = 1$. For each combination in positive attribute sub-space, its validity is tested. If the positive attribute combination is not valid, negative attribute sub-space is searched to form a valid combination composed of both positive and negative attributes. Thus the search space size for positive attribute sub-space is $\sum_{l=0}^{p} C_l^p = 2^p$. Even though this approach reduces the complexity in most cases, the worst case gives the same complexity as the generic tree search. If $p = n$ (i.e., no negative attributes), the search space size is 2^n. If every positive attribute combination needs negative attribute search, the search space will be

$$2^p \cdot 2^q = 2^{p+q} = 2^n. \tag{4.15}$$

The OAS reduces the cost dramatically due to its search structure and heuristics. The OAS composed of 3 phases: contribution scoring, sorting attributes according to

the scores and rule searching. Contribution scoring costs $O(n)$. Sorting phase costs $O(n \log n)$ with well-known sorting algorithms such as heap sort algorithm. Once attributes are sorted, searching the best rule costs $O(l)$ where $1 \leq l \leq n$. The l is the smallest length of valid rules to be searched. When b is greater than 1, the search starts from length l combinations, eliminating $\sum_{k=0}^{l-1} C_k^n$ combinations from search space. Even when searching C_l^n combinations, it reduces a large part of search space according to validity and weight sum of each combination, which is explained in the previous sections. Thus complexity of OAS is :

$$O(n \log n) \leq O(OAS) \leq O(n \log n) + O(C^n_{\lceil \frac{n}{2} \rceil}). \tag{4.16}$$

In summary, worst-case complexities of three search algorithms for any b are

$$\begin{array}{ccccc} & & O(EXHS) & = & O(2^n) \\ O(2^{n-1}) & \leq & O(TRS) & \leq & O(2^n) \\ O(n \log n) & \leq & O(OAS) & < & O(2^{n-1}). \end{array} \tag{4.17}$$

4.6 Comparison with Related Work

The primary differences between the OAS and other decompositional algorithms are twofold: (1) completeness and soundness of the ruleset generated; (2) computational complexity. Like KT algorithm, the OAS is intended to translate a trained neural network classifier into a rule-base classifier with equivalent performance, whereas some other algorithms (e.g., Gallant [10]) only extract a partial set of rules (or a single rule) which are then conjunction rather than in replacement of the network. The OAS generate a set of valid and maximally-general rules and the set covers a problem domain completely by an appropriate setting of parameter values. Secondly, the OAS is computationally more efficient than other algorithms. KT algorithm uses the separation of positive and negative attribute space and complexity control parameters in order to reduce the complexity, but its worst case complexity is still 2^n which is same as that of other tree-based search algorithms. The OAS's complexity is loosely upper-bounded by 2^{n-1} (i.e., $O(OAS) < 2^{n-1}$) for any case.

4.7 Experimental Results

4.7.1 Performance Evaluation

Performance evaluation of the OAS algorithm can be performed with different aspects:

1. Performance comparison on other rule extraction algorithms, which can be accomplished by the following criteria:

   (a) Validity and generality of generated rules,

   (b) Completeness of a ruleset,

   (c) Size of a ruleset and

   (d) Computational complexity.

   Computational complexity comparison is accomplished analytically. CPU running time comparison is meaningless because it is dependent on coding skills and hardware performance.

2. Performance comparison between rueset A and B in Figure 4.3:

   (a) Classification accuracy on test data sets (i.e., unseen data sets). Prediction capability of the classifiers is evaluated.

   (b) Classification accuracy on the domain data set. Comprehension capability of the classifier is evaluated.

   (c) Ruleset size.

   The ruleset B (in Figure 4.3) is extracted from the network and, thus, its performance is totally dependent on the network's performance. Therefore, performance comparison should include the network's performance.

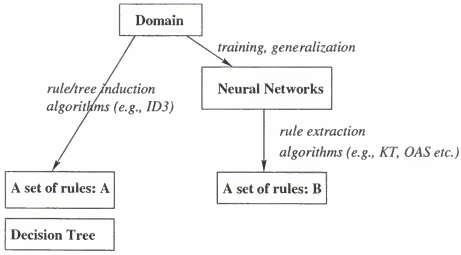


Figure 4.3. Performance evaluation of different classifiers.

Shavlik et al. [26] describes experimental comparison between ID3 symbolic learning algorithms and backpropagation neural learning algorithms. Their experimental results show that backpropagation neural learning algorithm performs slightly better than the ID3 in terms of classification accuracy on new examples, but takes much longer to train. However, backpropagation can work significantly better on data sets containing numerical data. It also occasionally outperforms ID3 when given relatively small amounts of training data and when examples are noisy or incompletely specified [26]. Hence, instead of performance comparison between the ruleset A and B (i.e., in Figure 4.3), rule extraction algorithms are evaluated by comparing predictive accuracy of the extracted ruleset with that of the network.

The OAS algorithm is proved to generate valid and maximally-general rules. Thus, Performance evaluation of the OAS algorithm in this section is accomplished in two aspects:

1. The OAS is compared with other related decompositional algorithms of rule extraction in terms of computational complexity. The complexity comparison is performed analytically in the section 4.5.

2. Empirical experiments are performed in this section. Performance comparison is performed between a neural network classifier and the ruleset (i.e., a ruleset B in Figure 4.3). Cross-validation is used to evaluate predictive accuracy of the classifiers. That is, an available data set is partitioned into k separate sets of approximately equal size. One set is used as a training set and the others as a test set. A neural network is trained with the training set and a ruleset is extracted from the trained network. Then, the network and ruleset are evaluated on the same test set.

The OAS algorithm is applied to four different well-known domains: XOR, Iris, Hypothyroid and Promoter. Performances of individual rules and rulesets are evaluated. Performance of an individual rule is evaluated by *coverage* and *accuracy* defined by:

$$\begin{aligned} coverage &= \frac{pos + neg}{total} \text{ and} \\ rule\ accuracy &= \frac{pos}{pos + neg} \end{aligned}$$

where *pos* is the number of positive instances, *neg* is the number of negative instances, and *total* is the number of instances in the test set. Note that $(pos + neg)$ is the number of instances covered by the rule, and thus $total = pos + neg + notcovered$ where the *notcovered* is the number of instances not covered by the rule. Performance of a ruleset is evaluated by the following:

$$\begin{aligned} coverage &= \frac{pos + neg}{total}, \\ confidence &= \frac{pos}{pos + neg} \text{ and} \\ classification\ accuracy &= \frac{pos}{total}. \end{aligned}$$

The *accuracy* of a ruleset is defined differently than that of a single rule. It is the performance of a ruleset classifier over the complete domain.

4.7.2 XOR problem

Two input attributes, four hidden units and one output unit are used. Nodes are fully connected between layers. Intermediate rules extracted from each hidden and output nodes and rewritten final rules are in Table 4.3.

4.7.3 Iris Domain

The Iris domain involves 4 continuous-valued attributes and 3 concepts. Each continuous-valued attribute is discretized to three interval attributes, producing a total of 12 binary input attributes. A three-layered neural network (12-4-3) is configured. Two experiments are performed: (1) comprehensive capability is evaluated by looking at how well training data instances are translated into a small set of concise rules, and (2) prediction capability is evaluated by looking at how well the extracted ruleset classifies unseen data instances, compared with the network.

A network is trained with a data set of 150 instances (50 instances for each concept). Training accuracy of the network is 98.67%, with only two instances misclassified. The OAS (Ordered Attribute Search) algorithm is applied to the trained network and nine rules are obtained (Table 4.4). Performance (coverage and accuracy) of the ruleset and each individual rules are evaluated over the 150 instances and are listed in Table 4.6 and Table 4.5. The ruleset covers 148 instances (i.e., 2 not-covered instances) of which only 2 mis-classifications are produced.

For prediction accuracy evaluation, the set of 150 instances is divided into 2 sets with 75 instances each. A network1 is trained with a training set and evaluated with a test set. A ruleset1 is generated from the network1 and evaluated with the test set. A network2 and a ruleset2 are evaluated in the same way. Prediction accuracy of the networks is illustrated in Figure 4.7, each network having one mis-classification on its test set. Figure 4.8 shows the prediction accuracy of the extracted rulesets. Ruleset1

I0 and I1 then H0 -0.827
not I1 then H0 0.780
not I0 then H0 0.785

I1 and I0 then H1 -0.787
not I1 and not I0 then H1 0.778

I0 and I1 then H2 -0.847
not I1 then H2 0.805
not I0 then H2 0.819

I0 then H3 -0.980
I1 then H3 -0.980
not I1 and not I0 then H3 0.937

not H2 and not H0 then O0 -0.966
H3 then O0 -0.974

not H3 and H2 and H1 then O0 0.885
not H3 and H2 and H0 then O0 0.992

I1 and not I0 then O0 0.992
I0 and not I1 then O0 0.992

not I1 and not I0 then O0 -0.974
I0 and I1 then O0 -0.966

Table 4.3. The intermediate rules and rewritten final rules for XOR problem

```
R1. (petal-length<=2.7) and (3.1<septal-width)          then setosa 0.952
R2. (petal-length<=2.7) and (petal-width<=0.7)          then setosa 0.845
R3. (petal-width<=0.7) and (3.1<septal-width)
    and (septal-length<5.4)                             then setosa 0.845

R4. (2.7<petal-length<=5.0) and (3.1<septal-width)
    and (5.4<=septal-length<6.3)                        then versicolor 0.992
R5. (2.7<petal-length<=5.0) and (0.7<petal-width<=1.6) then versicolor 0.992
R6. (2.7<petal-length<=5.0) and (6.3<=septal-length)
    and not (septal-width<=2.8)                         then versicolor 0.992

R7. (5.0<petal-length) and (1.6<petal-width)            then viginica 0.969
R8. (5.0<petal-length) and (septal-width<=2.8)          then viginica 0.969
R9. (1.6<petal-width) and (septal-width<=2.8)           then viginica 0.969
```

Table 4.4. Iris: Nine individual rules.

Rule	Size	Coverage(%)	Accuracy(%)
R1	2	30.0	100
R2	2	50.0	100
R3	3	28.0	100
R4	3	2.0	100
R5	2	45.5	98.9
R6	3	10.0	100
R7	2	39.5	98.7
R8	2	12.5	96
R9	2	16.5	96.97

Table 4.5. Iris: Performances of individual rules.

covers 73 instances (i.e., two uncovered instances) of which only 1 mi-classification is produced, and accuracy of ruleset2 is equivalent to that of the network2.

Ruleset Size	Coverage(%)	Confidence(%)	Classification(%)
9	98.67	98.65	97.33

Table 4.6. Iris: Comprehension accuracy of a ruleset.

	Prediction(%)
network1	98.7
network2	98.7

Table 4.7. Iris: Prediction accuracy of networks.

	Ruleset Size	Coverage(%)	Confidence(%)	Prediction(%)
ruleset1	11	97.3	98.6	96.0
ruleset2	7	100.0	98.7	98.7

Table 4.8. Iris: Prediction accuracy of rulesets.

4.7.4 Hypothyroid Disease Domain

The hypothyroid disease data set (provided by the Garavan Institute of Sydney, Australia) involves 2 concepts (hypothyroid and negative) and 26 variables: 7 continuous-valued and 19 binary attributes. The data set contains 3163 instances: 151 hypothyroid instances and 3012 negative instances. Some instances contain several missing values in their variables.

Since the data set involves continuous-valued variables, missing values and unbalanced a priori probabilities between the two concept instances, preprocessing is accomplished before experiments are performed. At first, the instances that include missing values in attribute TSH, TT4 and FTI are filtered out, leaving 2694 instances (150 hypothyroid, 2544 negative). To collect the same number of instances for each concepts, 150 negative instances are selected randomly from the 2544 negatives. Thus, a data set of 300 instances (150 hypothyroids and 150 negatives) is obtained for experiments. The data set is divided into two different sets (Set1 and Set2), each containing 150 instances (75 hypothyroids, 75 negatives). One set is used for training and the other set for testing. Among the 26 variables, 24 variables are used for this experiment because variable TGB includes too many missing values (91.8% missing). Thus two variables related to TGB (TGB and TGB measured) are excluded in this experimental data set. Six continuous-valued attributes are discretized to several binary attributes according their interval distributions, resulting in total 52 binary attributes.

A 3-layered neural network is configured (52-5-1) and trained with a training set. Rules are extracted from the trained network using the OAS algorithm and evaluated over training and testing set. Performances of neural networks and extracted rulesets are listed in Table 4.9. Neural network 1 is the one trained on Set1, and network 2 is trained on Set2. Ruleset1 is a set of rules extracted from neural network 1 and

	Classification(%)	
	set1	set2
Neural Network 1 (52-5-1)	98.7	96.0
Neural Network 2 (52-5-1)	97.3	98.7

	Coverage(%)		Confidence(%)		Classification(%)	
	set1	set2	set1	set2	set1	set2
Ruleset1 (7 rules)	75.33	64.7	100	98.97	75.33	64.03
Ruleset2 (8 rules)	88	87.3	100	99.24	88	86.64

Table 4.9. Hypothyroid: Performance of two rulesets extracted from two neural networks

```
R1. (FTI<70) and (TSH>=7.3) and not (60<age<=80)      then hypothyroid 0.906
R2. (FTI<70) and (TSH>=7.3) and (Female)              then hypothyroid 0.906
R3. (FTI<70) and (TSH>=7.3) and (TT4<=68)             then hypothyroid 0.906
R4. (FTI<70) and (TSH>=7.3) and (0<T3<=1)             then hypothyroid 0.886

R5. (FTI>=70) and (TSH<7.3) and (Male)
    and (on_thyroxine=f)                              then negative 0.946
R6. (FTI>=70) and (TSH<7.3) and (TT4>68)
    and (on_thyroxine=f)                              then negative 0.946
R7. (FTI>=70) and (TSH<7.3) and (1<T3<=2)             then negative 0.946
R8. (FTI>=70) and (TSH<7.3) and (thyroid_surgery=f)
    and (on_thyroxine=f) and not (40<age<=60)         then negative 0.946
```

Table 4.10. Hypothyroid: Eight individual rules in ruleset2.

ruleset2 is from network2. Individual rules in ruleset2 are listed in Table 4.10 and their performance (coverage and accuracy) are in Table 4.11.

Rule	Size	Coverage(%)	Accuracy(%)
R1	3	29.3	100
R2	3	34.7	100
R3	3	46.7	100
R4	3	26.7	100
R5	4	15.3	100
R6	4	39.3	100
R7	3	19.3	100
R8	5	30.7	100

Table 4.11. Hypothyroid: Performances of individual rules in ruleset2. Evaluated on test set(set1).

4.7.5 Promoter Domain

The promoter domain includes 2 concepts (promoter and non-promoter) and 57 multi-valued attributes. Each multi-valued attribute is discretized to four binary attributes, each binary attribute representing each value. Thus, total 228 binary input attributes are involved. Note that the binary value attributes of a multi-valued attributes are mutually-exclusive. The data set contains 106 instances: 53 promoter and 53 non-promoters.

A three-layered neural network is configured (228-4-1) and trained on the 106 instances with 100% training accuracy. Seven positive (promoter) rules are extracted. For negative (non-promoter) instances, a default rule is used, which states that *"if an instance is not covered by promoter rules, then it is non-promoter"*. Therefore, coverage of a ruleset is always 100%. Table 4.12 lists the 7 positive rules extracted and Table 4.14 and Table 4.13 show performances of the ruleset and individual rules. This experiment shows the translation of training samples into a small set of concise rules easily understood by users, which is a comprehension capability.

Second experiment is to investigate a prediction capability. The set of 106 instances is divided into two subsets with 53 instances each. A neural network is trained with one set and evaluated with the other set. A rule set is generated from the trained neural network and evaluated with the other set containing unseen instances. Two neural networks are trained with 100% accuracy and evaluated with a test data set. Prediction accuracy of the two networks is listed in Table 4.15. Table 4.16 illustrates prediction accuracy of the two rulesets generated from the networks. Note that coverage is always 100% because a default disconforming rule is used.

```
R1. @-34 "gxxa"                , then promoter 0.841
R2. @-36 "txxxxa"              , then promoter 0.841
R3. @-33 "axa"                 , then promoter 0.841
R4. @-34 "ga"                  , then promoter 0.841
R5. @-36 "txg"                 , then promoter 0.841
R6. @-36 "ttxa"                , then promoter 0.874
R7. @-36 "txxa" and @-5 "c"    , then promoter 0.874
Default rule. If not promoter, then non-promoter
```

Table 4.12. Promoter: Seven individual rules.

Rule	Size	Coverage(%)	Accuracy(%)
R1	2	16.9	94.4
R2	2	17.9	89.5
R3	2	11.3	83.3
R4	2	23.6	88.0
R5	2	32.1	94.1
R6	3	16.0	88.2
R7	3	11.3	91.7
Default rule	1	45.28	91.7

Table 4.13. Promoter Domain: Performances of individual rules.

Ruleset Size	Coverage(%)	Confidence(%)	Classification(%)
7	100.0	87.7	87.7

Table 4.14. Promoter Domain: Performances of ruleset.

	Prediction(%)
network1	90.6
network2	71.7

Table 4.15. Promoter: Prediction accuracy of networks.

	Ruleset Size	Coverage(%)	Confidence(%)	Prediction(%)
ruleset1	5	100.0	83.02	83.02
ruleset2	5	100.0	83.02	83.02

Table 4.16. Promoter: Prediction accuracy of rulesets.

CHAPTER 5
GA-BASED RULE EXTRACTION IN QUANTITATIVE DOMAIN

5.1 Introduction

In the previous chapters, rule extraction in qualitative (i.e., discrete attribute) domains has been investigated. Rule extraction becomes much more difficult when it comes to quantitative domains where each attribute is continuous-valued. Simple, but the most common way to handle quantitative domain is to discretize each continuous-valued attribute into several binary attributes and apply to the existing rule extraction techniques used for the binary attributes. But the discretization method may involve coding biases and may require a priori knowledge about the continuous-valued attribute to be divided into a certain number of ranges.

The neural network can be viewed as a function mapping input instances to classes. For a 3-layered feedforward network, it can be represented as

$$y_d(x_{i,i=1\ldots n}) = f(\sum_j w_{jd} f(\sum_i w_{ij} x_i - \theta_i) - \theta_j). \tag{5.1}$$

Input space is divided by each class according to its $y_k(X)$ where k is an index for each class and X is an input space. Consider the space covered by the class C_d. The issue is how to generalize the space with an understandable language, for example, a set of rules. If the attributes were binary, it searches each discrete points to verify its validation. For continuous-valued attributes, however, an arbitrary shaped space is to be represented in a simple form. Without loss of generality, each attribute values are normalized to the range of [0.0, 1.0]. A rule with conjunctive attributes whose value is a certain range between 0 and 1 forms a squared space at the dimension of input space. The problem is how to search the right range values for each attributes

so that the squared space covers the original class space as much as possible. Since the search space is continuous-valued space, any kind of exhaustive search would be very expensive in terms of its computation. In this section, genetic algorithm method is proposed, in which genetic search is applied to find the range values of attributes that maximize its representative space.

5.2 Genetic Algorithm

Genetic algorithms are used for optimization and learning based on the mechanism of genetic evolution. A genetic algorithm consists of five components:

1. A chromosomal representation of solutions
2. initialization of the population of chromosomes
3. A fitness function to evaluate solutions
4. Genetic operators to alter the composition of chromosomes during reproduction, e.g., crossover and mutation.
5. Parameter setting for the algorithm, e.g., population size and probabilities of applying genetic operators.

The population evolves to contain better individuals and finally converges to globally optimal (or near optimal) results. Let $m(H,t)$ denote the number of chromosomes of a particular configuration H at time t. Suppose during reproduction, a chromosome is copied according to its fitness, and the old population is replaced by the new population. Then, we may estimate the number at time $t+1$ by

$$m(H,t+1) = m(H,t)\frac{f(H)}{\overline{f}}$$

where $f(H)$ is the fitness of H and $\overline{f}$ is the average fitness of the entire population [11]. Starting at $t=0$ and assuming stationary fitness values, we obtain the equation

$$m(H,t+1) = m(H,0)(\frac{f(H)}{\overline{f}})^t.$$

Thus, it is clear that the numbers of favorable configurations will increase exponentially. A simple genetic algorithm proceeds as follows:

1. Start with a randomly generated population of n l-bit chromosomes.
2. Calculate the fitness $f(x)$ of each chromosome x in the population.
3. Repeat the following steps until n offsprings have been created:
   (a) Select a pair of parent chromosomes from the current population.
   (b) Cross over the pair at a randomly chosen point to form two springs.
   (c) Mutate the two offsprings at each locus and place the resulting chromosomes in the new population.
4. Replace the current population with the new population.
5. Go to step 2.

5.3 GA-Model

GA-based model is composed of encoding, crossover, mutation, selection and evolution.

5.3.1 Encoding

A chromosome is composed of genes. Each gene represents each continuous-valued attribute, composed of two values: lower bound and upper bound. Range of each input attribute is defined by the lower and upper bounds. For example, a continuous-valued attribute $x1$ defined as $0.2 < x1 < 0.7$ is encoded to a gene $[0.2, 0.7]$. Figure 5.1 illustrates the encoding mechanism. This chromosome has 3 genes and each gene has its lower and upper bound values. Each gene represents an input attribute and each lower or upper bound value represents a range bound of the attribute. Gene2 (i.e. attribute2) ranges in $[0, 1]$ which means *don't care* attribute.

5.3.2 Crossover

One-point crossover and uniform crossover are used. Each gene is a basic unit for crossover. Thus the values of lower bound and upper bound of a gene are not cut by crossover operator.

5.3.3 Mutation Based on Fitness

Mutation occurs by the mutation probability which is normally very low (for example, 0.01 or 0.1). Mutation on a binary string of a chromosome is simple: 0 becomes 1 and vice versa. However, mutation of a continuous-valued bits of a chromosome is not simple. Once a gene is decided to be mutated, a value is picked to replace the gene. The followings should be considered when the value is picked:

1. Variance of values to be picked.
2. Probability of values to be picked.
   (a) Uniform mutation. Any value within variance can be picked with equal probability.
   (b) Gaussian mutation Values within variance is picked according to Gaussian distribution.

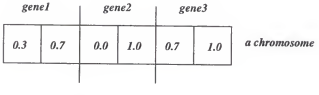


Figure 5.1. Encoding

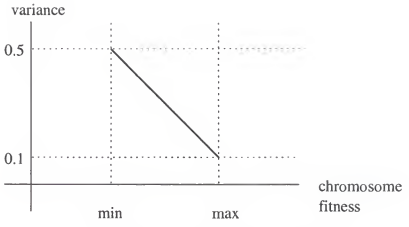


Figure 5.2. Variance of mutation

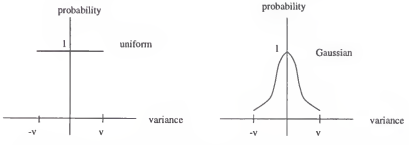


Figure 5.3. Probability of mutation values within variance

The variance is determined by fitness of chromosomes. A chromosome with high fitness is subject to a small variance of mutation while the one with low fitness is subject to larger variance of values to be picked for mutation. Figure 5.2 illustrates the simple method to determine the variance. Thus the chromosome with lowest fitness is subject to the mutation of the range $[\pm 0.0, \pm 0.5] = [-0.5, +0.5]$.

Once a variance is obtained, we need to consider which value to pick within the variance. For example, given a variance $[-0.5, 0.5]$, uniform mutation picks any value within the range randomly while Gaussian mutation picks the value according to Gaussian distribution which prefers small mutation values. Figure 5.3 illustrates the probability within variance. Let us say that a is a gene value to be mutated and m is the one picked from variance range. The mutated gene value a_{new} is defined as:

$$a_{new} = a + m.$$

5.3.4 Fitness Calculation

A fitness of an individual chromosome is determined by the following two values:

1. *Range*, an average of $(high - low)$ values of genes in the chromosome

2. *Act*, the lowest activation value that the chromosome can give to the neuron

The Range is used because we want the range of a gene is as large as possible so that a space defined by the chromosome is as big as possible. If a Range of a gene is maximum, that is, $[0.0, 1.0]$, the the gene (i.e. attribute in a rule) is considered *don't care* attribute.

The Act is an activation value of the neuron according to gene values in a candidate chromosome. What we want to find is the lowest activation value over the space defined by the gene values of a candidate chromosome. An activation is cut off at a certain threshold: if an activation gives a higher value, the value is reset to the threshold. The effect of this scheme is to restrict the evolution converging to activation 1 which results in deteriorating Range. We want to find the chromosome whose activation is above a certain threshold and that defines space as large as possible. At the example shown in figure 5.4, all the activation above 0.75 is reset to 0.75. The value 0.75 acts as a validity threshold used in decompositional methods (i.e., KT or OAS). Then, the range of activation is $[0.0, 0.75]$ and this range is normalized to $[0, 1]$.

Fitness is calculated as follows:

$$Fitness = \alpha \cdot Range + \beta \cdot Act$$

where α and β are credit constant.

5.4 Rule Search

An input space is divided by classes. Sometimes spaces covered by one class are disjoined (e.g. XOR problem). Hidden nodes in a neural network is known to detect

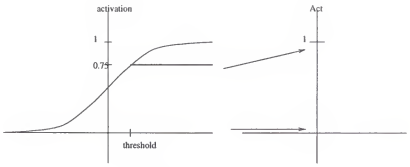
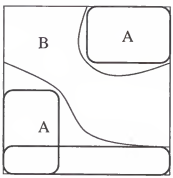
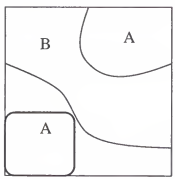


Figure 5.4. Act



Three chromosomes for class A cover nonlinear and disjoint spaces of . class A



One chromosome for class A is not enough to cover the spaces of A.

Figure 5.5. Correct number of rule chromosome covers nonlinear or disjoined boundaries between classes in a problem space.

those disjoined spaces. Figure 5.5 illustrates the importance of correct number of hidden nodes.

Let us take a look at a simple XOR domain problem. Two hidden nodes are used for this experiment. Figure 5.6 illustrates the trained neural network with connection weights and threshold. For GA evolution, 1-point and uniform crossover operators are used and uniform mutation operator is used. Pool size is four and initial chromosomes are defined randomly.

Figure 5.7 and 5.8 illustrate the spaces defined by chromosomes for hidden node 0 and 1, respectively. As seen, the squared space covers reasonably large valid area.

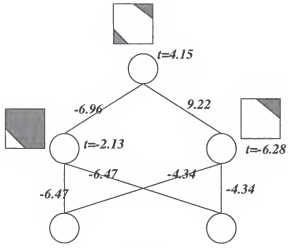


Figure 5.6. NN that is trained on XOR problem. t is a node threshold.

Each chromosome is now decoded into if-then rules as follows:

$$\begin{array}{llll} if & (0.0 < x1 < 0.12) & and & (0.0 < x2 < 0.13), \quad \text{then } h0 \\ if & (0.14 < x1 < 1.0) & and & (0.15 < x2 < 1.0), \quad \text{then not } h0 \\ if & (0.0 < x1 < 0.67) & and & (0.0 < x2 < 0.64), \quad \text{then } h1 \\ if & (0.68 < x1 < 1.0) & and & (0.66 < x2 < 1.0), \quad \text{then not } h1. \end{array} \tag{5.2}$$

Once hidden rules are extracted, output layer rule extraction and rewriting procedures are performed. Output layer rule extraction is same as the one used for discrete cases. For XOR domain case, output rules are

$$\begin{array}{llll} if & \text{not } h0 & and & h1, \quad \text{then } y \\ if & h0, & & \text{then not } y \\ if & \text{not } h1, & & \text{then not } y. \end{array} \tag{5.3}$$

Rewriting procedure is similar to the one used for discrete cases except few things. For negative rules in ruleset 5.3, rewriting replaces each hidden unit attributes by corresponding hidden rules in 5.2 since each rule contains only one attribute. For a rule that has a conjunctive hidden attribute like the positive rule at 5.3, however, their corresponding hidden rules are combined and each attribute ranges are adjusted. The positive rule in ruleset 5.3 is rewritten to

$$\begin{array}{llll} if & (0.14 < x1 < 1.0) & and & (0.15 < x2 < 1.0) \text{and} \\ & (0.0 < x1 < 0.67) & and & (0.0 < x2 < 0.64), \quad \text{then } y \end{array}$$

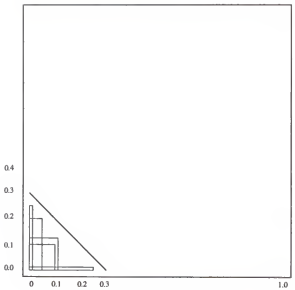


Figure 5.7. Hidden unit 0

and its attribute ranges are combined to

$$\text{if } (0.14 < x1 < 0.67) \text{ and } (0.15 < x2 < 0.64), \text{ then } y. \tag{5.4}$$

5.5 Experimental Results

Iris domain involves four continuous-valued attributes (A, B, C and D) and three classes (Y0, Y1 and Y2). Without loss of generality, the continuous values are normalized into the ones in range $[0, 1]$. A 4-4-3 feedforward neural network is used to train 150 instances in the Iris domain with accuracy of 97.3% (4 instances misclassified). For each hidden nodes, the GA-based model extracts several rules and the best rules for each hidden units are as follows:

$$\begin{array}{llll}
if\ A[0.00, 0.75] & B[0.43, 1.00] & C[0.00, 0.21] & D[0.00, 0.05], \text{ then } H0 \\
if & B[0.00, 0.95] & C[0.58, 1.00] & D[0.52, 1.00], \text{ then not } H0 \\
if\ A[0.29, 1.00] & & C[0.00, 0.58] & D[0.00, 0.52], \text{ then } H1 \\
if\ A[0.06, 0.83] & & C[0.61, 0.96] & D[0.82, 1.00], \text{ then not } H1 \\
if\ A[0.62, 1.00] & B[0.00, 0.49] & C[0.77, 1.00] & D[0.60, 1.00], \text{ then } H2 \\
if & B[0.32, 1.00] & C[0.00, 0.13] & D[0.00, 0.67], \text{ then not } H2 \\
if\ A[0.54, 1.00] & B[0.00, 0.44] & C[0.76, 1.00] & D[0.74, 1.00], \text{ then } H3 \\
 & C[0.00, 0.35] & D[0.00, 0.46], & \text{then not } H3.
\end{array} \tag{5.5}$$

Rules for output units are extracted using the OAS:

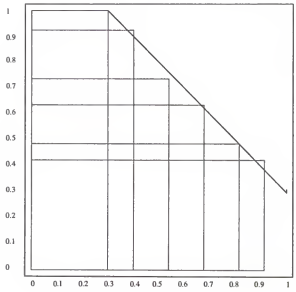


Figure 5.8. Hidden unit 1

```
R1. (septal-length<7.52) and (2.95<septal-width) and
    (petal-length<2.24) and (petal-width<0.22),          then setosa
R2. (2.7<septal-width<4.09) and (4.42<petal-length) and
    (1.35<petal-width<1.71),                             then versicolor
R3. (7.33<septal-length<7.65) and (septal-width<3.08) and
    (5.54<petal-length<6.66) and (2.07<petal-width),     then viginica
R4. (7.21<septal-length) and (septal-width<2.97) and
    (5.48<petal-length<6.66) and (2.07<petal-width),     then viginica
```

Table 5.1. Quantitative Iris: four rules.

```
H0 and not H3 then Y0 0.627
H0 and not H2 then Y0 0.795

not H0 and not H3 and not H2 then Y1 0.756
not H0 and H1 then Y1 0.766

not H1 and H3 and H2 then Y2 0.885
not H1 and not H0 and H2 then Y2 0.885
not H1 and not H0 and H3 then Y2 0.975.
```

After rewriting and de-normalization of values, final rules are listed in Table 5.1. Performance of rules are shown in Table 5.2 and 5.3. As expected, the coverage is low. This is because of the property if-then rules. Classificaton boundary is non-linear and the if-then rules can not fit completely the space of non-linear boundaries.

Rule	Size	Coverage(%)	Accuracy(%)
R1	4	18.7	100
R2	3	10.0	86.7
R3	4	1.33	100
R4	4	4.67	100

Table 5.2. Quantitative Iris: Performances of individual rules.

Ruleset Size	Coverage(%)	Confidence(%)	Classification
4	34.67	96.15	33.34

Table 5.3. Quantitative Iris: Performances of a ruleset.

CHAPTER 6
CASE STUDY: DATA MINING IN TELECOMMUNICATIONS INDUSTRY

6.1 Introduction

In this section, the neural network approach is applied to data mining problems in the real-world environment. The purpose of data mining is to provide useful, understandable and compressed information to users. A data mining process is usually performed on data warehouses or departmental data marts. Data warehouse provides multidimensional databases and more advanced query tools such as OLAP (On Line Analytical Processing). Users want to make complex queries of multidimensional databases, and require fast and detailed summaries in response to such queries, which SQL - the standard for database queries - is not capable of performing. An alternative framework, known as OLAP (On Line Analytical Processing) has become popular which enables basic analysis and fast reporting. Although it allows complex queries and performs basic analysis such as aggregates, it does not automatically extract knowledge.

Data mining technology is based primarily on statistics. Techniques such as regression, clustering and factor analysis have been used for many years. Thanks to increases in the processing power of modern computers and the reduction in their price, the techniques have been expanded to include machine learning tools such as decision trees, neural networks, etc. In addition, advances in information theory and understanding of probabilistic inference have enabled many new techniques to be invented. Finally, the improvements in the software engineering and graphical visualization of data have also brought the power of data mining technology to the user.

Most of the real world data is partly structured, non-linear and noisy. To deal with the data, stochastic strategies and adaptive methods have been introduced. Neural Networks are good at processing noisy data and non-linear data. The problem with neural networks is that the data set is compressed into a set of connection weights, which is not easily understandable by human. Rule extraction strategies provide the capability to understand what the neural networks learned. The combination of neural networks and rule extraction methods is simulated on churn analysis, which is one of the most challenging data mining problems.

6.2 Churn Analysis

Telecommunications, banking or insurance companies (e.g. AT&T, MCI, GTE etc) maintain enterprise data warehouse and data mining processes for their business purposes, such as customer retention, database marketing, market analysis, target marketing etc. The following data are loaded to data warehouse:

Operational data Data from legacy systems, such as service order system, billing system and customer service system. The data includes call statistics, billing data, product and service data, complaint information etc.

Customer demographics Location, marital status, credit history, salary etc.

Combining those data from different sources, data warehouse provides a customer profile database and multidimensional view of the database. Users obtain information and business rules with the help of data mining tools. For example, marketing and sales users in telecommunications, banking and insurance companies will be interested in the following information from data warehouse:

1. Customer profile information in order to support effective account management and competitive analysis.

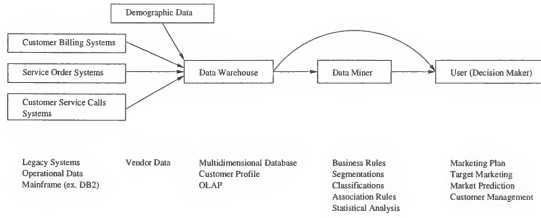


Figure 6.1. An example of decision making process in the telecommunications industry.

2. Customer defection (i.e., churn) analysis in order to keep their existing customers.

3. Market segmentation to decide what products to offer.

4. Analysis of customer behavior and patterns for specified marketing program development.

5. Targeting new customer.

6. Customer value evaluation to determine the customer's value on the bottom line.

7. Profitability: Determining the impact of changing products and services.

8. Business rules from historical data.

Figure 6.1 illustrates an example of data warehouse operation in the telecom industry.

In the last few years, the telecommunication industry has changed dramatically. Competition among long distance service providers has increased. Local carriers,

independent carriers and even foreign carriers have joined the competition over local and long distance services and wireless services. Due to the liberalization of the market place there is a serious need to understand customers. Given the increase in customer choice, there has been an increase in churn rates. Churn, or customer defection occurs when a customer switches providers for any reason, such as lower rates or better customer services. Churn in the telecommunications industry is an expensive problem. On average, the annual rate of churn is approximately 30% of total telecommunication customers in the US alone and the estimated cost of acquiring a new customer is approximately $400 per customer in wireless communications according to 1995 report from Digital Equipment Corp. It is clear that spending money to keep existing customers is more efficient than acquiring new customers. Until now, the industry has reacted by launching more aggressive marketing programs to acquire new subscribers. In some circumstances, these promotional schemes can lead to greater debt and fraud as they focus simply on the volume of customers rather than the right type of customers. To help reduce subscriber churn, organizations need to be able to identify those who are likely to churn and the reasons why - well before the subscriber takes any action. By predicting which customer is likely to churn and when it will happen, the company can offer the customer new incentives to stay. By understanding why a particular customer switches to another company, the company can provide different service products and change marketing program to satisfy the customer ahead of time. The "prediction" and "understanding" are two key tasks in churn analysis.

A neural network is an effective tool for prediction and it has been used successfully in credit card fraud detection. It also can be used to predict the likelihood of a customer churning. In this case study, neural network prediction and OAS interpretation are applied to churn analysis.

6.3 Churn Data Preparation

Churn data used in this case study is provided in the Stanford University MLC++ library of telecommunication customer churn information. The data set is artificial based on claims similar to the real world. This contains 3333 records with 20 fields and 1 class attribute. For this data set, 14.5% (485 records) of the population churned. The 21 attributes include the following fields:

1. State Code: discrete (51 different codes).
2. Account length in days: continuous.
3. Area code: continuous.
4. Phone number: discrete.
5. International plan: discrete (yes or no).
6. Voice mail plan: discrete (yes or no).
7. Number voice mail messages: continuous.
8. Total day time calls in minutes per month: continuous.
9. Total number of day time calls per month: continuous.
10. Total day time call charge per month: continuous.
11. Total evening time calls in minutes per month: continuous.
12. Total number of evening time calls per month: continuous.
13. Total evening time call charge per month: continuous.
14. Total night time calls in minutes per month: continuous.

15. Total number of night time calls per month: continuous.

16. Total night time call charge per month: continuous.

17. Total international calls in minutes per month: continuous.

18. Total number of international calls per month: continuous.

19. Total international call charge per month: continuous.

20. Number of customer service calls: continuous.

21. Churn status: discrete (false or true).

From the 20 input attributes, 9 attributes are used, excluding unimportant attributes such as state code, area code, phone number, number of calls per month and monthly charge. The data of the 7 continuous attributes in a training set are well normally-distributed.

The first thing to do is to discretize continuous attributes. In this case study, each range of continuous attribute is divided into a fixed number of binary attributes with equal sub-range size. Table 6.1 illustrates the selected attributes and their discretization. After the discretization, we have 31 binary input attributes.

6.4 Rule Extraction and Discussion

A neural network is configured as 31-6-1 and trained on the 3333 instances (85.51% churn instances and 14.49% non-churn instances). Classification accuracy of the trained network is 91.4%. There are 287 mis-classified instances: 72 for non-churn, and 215 for churn.

Table 6.2 illustrates performances of rulesets extracted with different parameter values of N1 and N2. As the parameter values increase, more rules are extracted and coverage increases. The difference between the performances of the neural network (91.4%) and a ruleset is believed to be caused by the coding bias of discretization.

Attribute	Max value	The # of intervals	Size of each interval
account length	243	5	61
international plan	no/yes	2	NA
voice mail plan	no/yes	2	NA
voice mail message	51	4	13
daytime call minutes	35000	4	8750
evening call minutes	36300	3	12100
night call minutes	39500	3	13161
international call minutes	2000	5	400
# of service calls	900	3	300

Table 6.1. Input attribute discretization. Total 31 binary input elements are used. The elements discretized from an attribute are mutually exclusive at rule combinations. Max value is the highest value of the attribute.

Since the confidences of rulesets are about the same, classification accuracy of a ruleset depends on its coverage.

A ruleset represents the instances in the data set. Consider the ruleset extracted with $N1 = 10$ and $N2 = 2$. The ruleset has only 31 rules, representing 91.3% of 3333 instances with confidence of 89%. In other words, 3044 instances are summarized by 31 rule instances, and 94364 bits (i.e., $3044 * 31$ bits) are reduced into 350 bits. In data mining aspect, a telecom company decision maker looks for a concise pattern that explains behavior of a large part of customers. If the pattern covers 50% of millions of customers with high confidence, the telecom company programs incentive plans or new services for the customers. Hence, more interest is on "a small set of concise rules," rather than "a large and complete set of rules." If generality (i.e., coverage) of a rule is large, it would be easier and more productive to program a special incentive plans or new services for the group of customers. Table 6.3 and 6.4 illustrates a ruleset and its performance for the case of $N1 = 3$ and $N2 = 2$. Rule R7 classifies 36.48% of customers into non-churn group with 90.13% of accuracy.

N1	N2	Ruleset Size	Coverage(%)	Confidence(%)	Classification(%)
2	1	5	19.8	87.1	17.25
2	2	5	21.3	87.6	18.66
3	1	7	39.0	90.2	35.18
3	2	8	40.5	90.4	36.61
4	1	14	40.7	89.4	36.39
4	2	15	42.3	89.5	37.86
5	1	20	41.9	88.8	37.21
5	2	20	41.9	88.8	37.21
6	1	17	44.9	89.0	39.96
6	2	17	46.7	88.3	41.24
7	1	18	52.7	89.5	47.17
7	2	19	55.4	88.4	48.97
8	1	19	62.8	88.8	55.77
8	2	20	65.5	87.9	57.57
9	1	22	72.9	88.9	64.81
9	2	23	74.8	88.0	65.82
10	1	30	89.4	89.8	80.28
10	2	31	91.3	89.0	81.25
11	1	39	82.4	88.9	73.25
11	2	40	84.8	88.1	74.71
12	1	40	95.4	88.9	84.81
12	2	39	96.6	88.1	85.1
13	1	41	95.4	88.9	84.8
13	2	40	96.6	88.1	85.1

Table 6.2. Performance of rulesets extracted with different parameter values: N1 and N2 are the number of rules (confirming or disconfirming rules) extracted from hidden layer and output layer, respectively. N1 and N2 are set by a user. Coverage is a percentage of the instances covered by the ruleset over a domain. Confidence is a percentage of positive instances over the covered. Classification accuracy is a percentage of the positive instances over the domain.

I6 and I27 and I30 and I13 then O0 1.000
I6 and I27 and I30 and I14 then O0 1.000
I6 and I27 and I30 and I4 then O0 1.000
I26 and I6 then O0 1.000
I6 and I27 and I16 then O0 1.000
I23 and I29 then O0 -0.972
I25 and I5 and I29 then O0 -0.972
I25 and I0 and I29 then O0 -0.972

Table 6.3. The 8 rules extracted with N1=3 and N2=2.

Rule	Coverage (number)	Accuracy(%)	churn
R1	0	NA	yes
R2	0	NA	yes
R3	0	NA	yes
R4	54	94.4	yes
R5	0	NA	yes
R6	74	91.89	no
R7	1216	90.13	no
R8	116	89.66	no

Table 6.4. Performance of individual rules extracted with parameter N1=3 and N2=2.

According to the rules extracted, in general, churning groups are the customers with "high international calls, international call plan and high customer service calls," and non-churning groups are those with "low international calls and low customer service calls."

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Summary

A neural network is a good model that estimates any smooth nonlinear function and adapts the function to the data used in training without any a priori assumptions. The model, however, is hidden in a black box, and can be used to predict, but not to understand the environment. Interpreting the neural networks in a language that is easily understood by users is essential to knowledge discovery and data mining applications. It allows the user to inspect the knowledge generated by the model and judge its usability for particular decisions.

First of all, recent research on rule extraction from neural networks has been surveyed. This dissertation explores many different ways to improve performance with KT-type rule extraction algorithms. Based on the algorithm, heuristics to control the ruleset complexity are studied. Then, neural network structure refinement is simulated using the algorithm. To reduce the search space and find a set of maximally general and valid rules, a new search algorithm, called the *ordered attribute algorithm*, is introduced. This algorithm generates a set of valid rules with much efficient time complexity compared to other algorithms. Finally, a case study of churn analysis in the telecommunications industry is studied. This case study shows how the rule extraction algorithm can be applied to data mining problems in the real-world situations.

Rule extraction from the network that is trained on quantitative domains is a difficult problem because of the unlimited size of search space. Extracting if-then rules can be viewed as discretizing a problem domain and, therefore, finding valid intervals

at each quantitative dimension. Searching for maximally-general and valid intervals is still an open problem. A GA-based approach is introduced in this dissertation to find the optimal intervals heuristically. More theoretical and systemic approaches to the problem have to be investigated.

7.2 Contributions

This dissertation has made several contributions to the state of the art in knowledge discovery models via neural networks. These contributions are described in some detail as follows:

1. The Ordered-Attribute Search (OAS) algorithm.
   (a) Computational complexity is reduced significantly, compared to existing algorithms.
   (b) It generates b best rules (i.e., valid and maximally general) while existing algorithms generate an arbitrary number of rules with confidences (or certainty factors) greater than a validity threshold.
   (c) It generates a complete set of valid rules while some other algorithms generate a partial set of rules.
2. Research and development of KT algorithm.
   (a) Implementation of KT-based rule extraction system with complexity control parameters in a multi-valued domain environment.
   (b) Experiments with network refinement.
3. Investigation of generality capability of a RBNN (Rule-Based Neural Network) in many different situations.
4. A GA-based approach to quantitative domain.

7.3 Problems and Future Work

Decompositional approaches are based on the assumption of boolean activation in hidden units. As described in section 3.4, it suffers the discontinuity problem. There has been some approaches using multi-valued activation instead of boolean activation [25], but it still does not solve the problem. This problem biases the exact translation of a neural network into a rule base, especially in the rewriting procedure of intermediate rules. Thus, one of interesting task for furure research would be the framework to deal with the continuous-valued activations in hidden units.

Analytic and systemic approach to quantitative domain is still an open research issue. If-then rule extraction on quantitative domains can be viewed as finding valid and maximally-general intervals at each dimension of the domain. Since the space defined by the intervals is rectangular, it is difficult for the if-then rules to cover non-linear boundaries in the domain. Knowledge representation that is appropriate for quantitative domains should be studied.

REFERENCES

[1] Robert Andrews, Joachim Diederich, and Alan Tickleu. "Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks". *Knowledge-Based Systems*, 8(6):373–389, 1995.

[2] Etienne Barnard and Elizabeth Botha. Back-propagation uses prior information efficiently. *IEEE Transactions on Neural Networks*, 4(5), September 1993.

[3] Etienne Barnard, R. Cole, and L. Hou. Location and classification of plosive consonants using expert knowledge and neural-net classifiers. *J. Accoust. Soc. Amer.*, 84, 1988.

[4] Mark Craven. *Extracting Comprehensible Models from Trained Neural Networks.* PhD thesis, University of Wisconsin - Madison, 1996.

[5] Mark Craven and Jude Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning: Proceedings of Eleventh International Conference*, San Francisco, CA, 1994.

[6] LiMin Fu. "Knowledge-Based Connectionism for Revising Domain Theories". *IEEE Transactions on System, Man, and Cybernetics*, 23(1):173–182, 1993.

[7] LiMin Fu. *Neural Networks In Computer Intelligence*, chapter 14: rule generation from neural networks. McGraw-Hill, Inc., 1994.

[8] LiMin Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(8), 1994.

[9] LiMin Fu and Hyeoncheol Kim. Abstraction and representation of hidden knowledge in an adapted neural network. unpublished, October 1994.

[10] Stephen I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2), February 1988.

[11] David E. Goldberg. *Genetic Algorithms.* Addison-Wesley, 1989.

[12] Hyeoncheol Kim. GA-based parameter setting for KT rule extraction. unpublished, November 1996.

[13] Hyeoncheol Kim and LiMin Fu. Generalization and fault tolerance in rule-based neural networks. In *Proceedings of IEEE International Conference on Neural Networks*, volume 3, pages 1550–1555, Orlando, FL, 1994.

[14] Igor Kononenko and Ivan Bratko. Information-based evaluation criterion for classifier's performance. *Machine Learning*, 6:67–80, 1991.

[15] A. Krogh and J.A. Hertz. "A Simple Weight Decay Can Improve Generalization". In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 950–957, San Mateo, CA, 1992. Morgan Kauffmann Publishers.

[16] J.K. Kruschke and J.R. Movellan. "Benefits of gain: Speeded learning and minimal hidden layers in back-propagation networks". *IEEE Transactions on systems, man and cybernetics*, 21(1), January, 1991.

[17] Hongjun Lu, Rudy Setiono, and Huan Liu. Neurorule: A connectionist approach to data mining. In *Proceedings of the 21st VLDB Conference*, pages 478–489, Zurich, Switzerland, 1995.

[18] M. Mozer and P. Smolensky. "Skeletonization: A Technique For Trimming The Fat From A Network Via Relevance Assessment". In D.S. Touretzky, editor, *Advances in Neural Information Processing System 1*, pages 107–115. Morgan Kaufmann Publishing, San Mateo, CA., 1989.

[19] S.J. Nowlan and G.E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

[20] Gregory Piatetsky-Shapiro. Discovery, analysis, and presentation of string rules. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery In Databases*, pages 229–248. AAAI Press/ MIT Press, CA, 1991.

[21] Michael Richard and Richard Lippmann. Neural network classifiers estimate bayesian a posteriori probabilities. *Neural Computation*, 3:461–483, 1991.

[22] Dennis Ruck, S. Rogers, M. Kabrisky, M. Oxley, and B. Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4), December 1990.

[23] Kazumi Saito and Ryohei Nakano. Rule extraction from facts and neural networks. pages 379–362, 198?

[24] Kazumi Saito and Ryohei Nakano. Medical diagnostic expert system based on pdp model. In *Proceedings of International Conference on Neural Networks*, volume 1, pages 255–262, San Diego, CA, 1988.

[25] Rudy Setiono and Huan Liu. Understanding neural networks via rule extraction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 480–485, Montreal, Canada, 1995.

[26] Jude Shavlik, Raymond Mooney, and Geoffrey Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143, 1991.

[27] Padhraic Smyth and Rodney M. Goodman. Rule induction using information theory. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery In Databases*, pages 159–176. AAAI Press/ MIT Press, CA, 1991.

[28] Padhraic Smyth and Rodney M. Goodman. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(4), August 1992.

[29] Sebastian Thrun. "Extracting Provably Correct Rules from Artificial Neural Networks". Technical Report IAI-TR-93-5, Institut for Informatik III Universitat Bonn, Germany, 1993.

[30] Sebastian Thrun. Extracting rules from artificial neural networks with distributed representations. In G. Tesauro, D.S. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing System 7*. Morgan Kaufmann Publishing, San Mateo, CA., 1995.

[31] Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1), October 1993.

[32] Geoffrey G. Towell, Jude W. Shavlik, and M.O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings AAAI-90*, pages 861–866, Boston, MA, 1990.

[33] Volker Tresp, Jurgen Hollatz, and Subutai Ahmad. Network structuring and training using rule-based knowledge. In C.L. Giles, S.J. Hanson, and J.D. Cowan, editors, *Advances in Neural Information Processing System 5*. Morgan Kaufmann Publishing, San Mateo, CA., 1993.

[34] Eric Wan. Neural network classification: A bayesian interpretation. *IEEE Transactions on Neural Networks*, 1(4), December 1990.

[35] David C. Wilkins and Bruce G. Buchanan. On debugging rule sets when reasoning under uncertainty. In *Proceedings AAAI-86 Fifth national conference on artificial intelligence*, volume 1, pages 448–454, Philadelphia, PA, 1986.

BIOGRAPHICAL SKETCH

Hyeoncheol Kim was born in Kang-Reung, Korea in 1964. He received a B.S. degree in computer science from Korea University, Seoul, Korea in 1988 and an M.S. degree in computer science from University of Missouri-Rolla in 1990. In the fall of 1992, he began the Ph.D. program in computer and information sciences at the University of Florida and plans to graduate in 1998. Since 1993, he has been a graduate research assistant in the areas of neural networks, rule extraction from neural networks, and data mining. He has also worked for GTE Data Services, Inc. in Tampa, Florida since January 1998. His research interests include neural networks, data mining, knowledge discovery, and data warehousing. He likes to spend his leisure time traveling and experiencing different cultures. His active involvement with the Korean Students Association includes being vice-president at the University of Florida and president at the University of Missouri-Rolla.